8 Phát triển chương trình

[Phạm vi câu hỏi]
Các ngôn lập trình (C, Java™),
Viết chương trình,
Môi trường phát triển
Các phương pháp kiểm thử,....

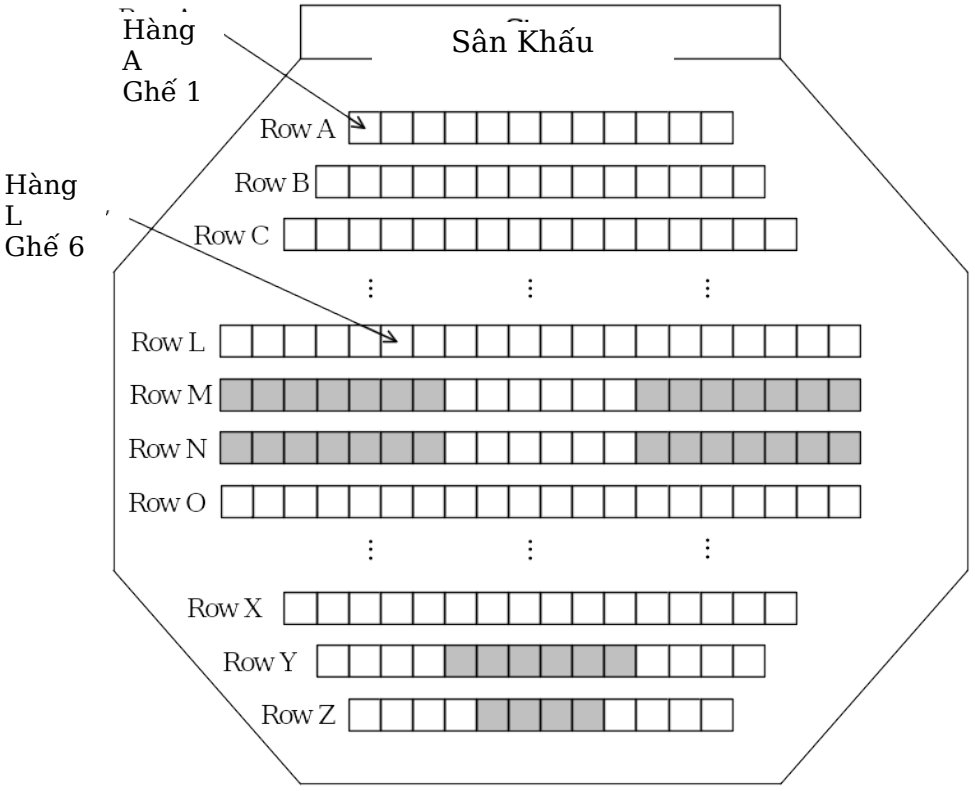
Câu hỏi 1 Lập trình C

Q1. Đọc mô tả chương trình và chương trình C dưới đây rồi trả lời các Câu hỏi con.

[Mô tả chương trình]

Đây là một chương trình đặt chỗ ngồi cho một phòng hòa nhạc. Chương trình nhận số lượng liên tiếp các chỗ ngồi cần được đặt trước làm dữ li vào, sau đó phân phối lại các chỗ ngồi, và trả về kết quả.

(1) Sự sắp xếp chỗ ngồi trong phòng hòa nhạc được thể hiện trong hình dưới đây:



Hình 8.1.1. Sắp xếp chỗ ngồi trong phòng hòa nhạc

(i) Có 26 hàng ghế được mã hóa từ A đến Z. Mỗi hàng có số lượng ghế khác nhau. Số ghế trong mỗi hàng được lưu trong mảng c theo thứ tự từ hàng A đến Z.

(ii) Số ghế bao gồm mã hàng ghế và một số. Số đó chỉ ra vị trí của ghế bắt đầu từ bên trái của mỗi hàng nhìn lên phía sân khấu. Ví dụ, ghế "hàng L, số 6" là ghế thứ 6 từ bên trái khi đứng đối diện với s khấu, trong hàng thứ 12 (hàng L) từ phía trên sân khấu.

(iii) Các ghế trong vùng tô đậm trong hình vẽ đã được đặt trước.

(2) Số lượng ghế liên tiếp đã được chọn sẽ được lưu vào tham số `number` và truyền vào chương trình.

(3) Chương trình bắt đầu tìm kiếm từ chiếc ghế ngoài cùng bên trái trong hàng đầu tiên (hàng A, số 1) khi đứng đối diện sân khấu, tìm số lượng các ghế liên tiếp được yên cầu trong cùng một hàng, và chọn cụm các ghế trống đầu tiên liên tiếp nhau mà nó tìm thấy. Nếu các ghế không thể được tìm thấy trong hàng A, chương trình sẽ tìm tuần tự từ ghế ngoài cùng bên trái trong hàng B, C... cho đến khi tìm thấy cụm các ghế trống đầu tiên liên tiếp nhau. Nếu số lượng các ghế liên tiếp nhau được yêu cầu không thể tìm được, chương trình sẽ xác định rằng các ghế không thể tìm được đó.

(4) Nếu số lượng yêu cầu của các ghế liên tiếp nhau được tìm thấy, chương trình trả về con trỏ trỏ đến cấu trúc số ghế (`SEAT`), cấu trúc này chứa mã hàng của các ghế, và số ghế tính từ ghế ngoài cùng bên trái (khi đứng đối diện với sân khấu). Nếu không tìm thấy, kết quả trả về con trỏ rỗng (`NULL`).

(5) Cấu trúc SEAT được mô tả như sau:

```
typedef struct {
    char  row;   /* mã hàng */
    int   no;    /* sô´      */
} SEAT;
```

(6) Trạng thái của mỗi ghế trong phòng hòa nhạc được lưu trong biến toàn cục `status`. Nếu ghế được tìm thấy, nó được đánh dấu là đã đặt trước.

`Status[i][j]=` { `' '`: Ghế trống.
{ `'R'`: Ghế đã có người.

Giá trị của chỉ số i trong khoảng từ 0 đến 25 tương ứng với các hàng từ A đến Z. Giá trị của chỉ số j trong khoảng từ 0 đến n-1 tương ứng với các số từ 1 đến n.

[Chương trình]

```
#define MAXNUM 30    /* Số lượng ghế tối đa trong một hàng*/
#define ROWNUM 26    /* Số lượng hàng */

typedef struct {
    char  row;       /* Mã hàng */
    int   no;        /* Số ghế */
} SEAT;

static char     rname[] = {"ABCDEFGHIJKLMNOPQRSTUVWXYZ"},
                status[ROWNUM][MAXNUM];
static int      cnum[ROWNUM] = {12, 14, 16, 18, 20, 22, 24, 26, 28,30, 30,
30, 30, 30, 30, 30, 30, 28, 26, 24, 22, 20, 18, 16, 14, 12};
static SEAT     empty;

SEAT *book_seat(int);

SEAT *book_seat(int number)
{
    int ridx, cidx, eidx, flg = 0;

    for (ridx = 0; ridx < ROWNUM; ridx++) {
        for (cidx = 0; cidx <= cnum[ridx] - number; cidx++) {
            if (status[ridx][cidx] == ' ') {
                [ A ];
                for (eidx = [ B ]; cidx < eidx; eidx--)
                    if (status[ridx][eidx] == 'R') {
                        flg = 0;
                        break;
                    }
                if (flg == 1) break;
            }
        }
        if (flg == 1) break;
    }

    if (flg == 0)
        return NULL;
```

```
    for (eidx = cidx + number - 1; cidx <= eidx; eidx--)
        status[ridx][eidx] = 'R';
    empty.row = [ C ];
    empty.no  = [ D ];
    return &empty;
}
```

Câu hỏi

Từ các nhóm câu trả lời dưới đây, chọn các câu trả lời đúng để điền vào trong ô trống [ ] trong chương trình trên.

Nhóm câu trả lời cho A:

a) `cidx++`
b) `cidx--`
c) `cidx += number`
d) `cidx += number - 1`
e) `flg = 0`
f) `flg = 1`

Nhóm câu trả lời cho B:

a) `0`
b) `cidx`
c) `cidx + 1`
d) `cidx + number`
e) `cidx + number - 1`
f) `number`
g) `number - 1`

Nhóm câu trả lời cho C:

a) `ridx`
b) `ridx + 1`
c) `ridx - 1`
d) `rname[ridx]`
e) `rname[ridx + 1]`
f) `rname[ridx - 1]`

Nhóm câu trả lời cho D:

a) `cidx`
b) `cidx + 1`
c) `cidx - 1`
d) `ridx + cidx`
e) `ridx - cidx`

Đáp án câu 1

Tìm kiếm và chọn các ghế trống liên tiếp trong một phòng hòa nhạc.

Đáp án đúng

A – f, B – e, C – d, D – b

Giải thích

Đây là câu hỏi tương đối dễ dàng tương tự như quá trình tìm kiếm một xâu kí tự được lưu trong một mảng 2 chiều. Khi bạn đọc chương trình, đặc biệt chú ý tới ba điểm sau: “vai trò của mảng đang được dùng”, “mục đích và nội dung của các biến đang được dùng” và “cấu trúc của các tiến trình”. Sau đó bạn có thể sẽ phải điền vào các ô trống trong quá trình đọc chương trình.

Trong phần giải thích này, trước tiên chúng ta sẽ cố gắng nắm được tổng quan của chương trình, thiết lập các đối tượng sẽ được trả lời, và sau đó tạo các giải thích chi tiết ở những nơi cần thiết. Chương trình liệt kê dưới đây là hàm "book_seat(") được viết bằng giả ngôn ngữ cộng vào đó số hiệu các dòng để giải thích và các dòng phụ trợ để giúp chúng ta thiết lập miền và cấu trúc phân cấp của từng khối tiến trình.

```
1  SEAT *book_seat(int number)
2  {
3     ○int ridx, cidx, eidx, flg = 0;
4
5     ■for (ridx = 0; ridx < ROWNUM; ridx++) {
6        ■for (cidx = 0; cidx <= cnum[ridx] − number; cidx++) {
7           ▲if (status[ridx][cidx] == ' ' ) {
8                    [   a   ];
9              ■for (eidx = [     b     ]; cidx < eidx; eidx--)
10                ▲if (status[ridx][eidx] == 'R') {
11                            flg = 0;
12                            break;
13             ■   ▼}
14             if (flg == 1) break;
15          ▼}
16       ■}
17       if (flg == 1) break;
```

```
18      }
19      if (flg == 0)
20          return NULL;
21      for (eidx = cidx + number - 1; cidx <= eidx; eidx--)
22          status[ridx][eidx] = 'R' ;
23      empty.row = [    c    ];
24      empty.no  = [    d    ];
25      return &empty;
26  }
```

Đọc lướt qua [Mô tả chương trình] của câu hỏi. Lướt qua toàn bộ [Chương trình], tập trung vào "mảng", "biến", và "cấu trúc tiến trình", bạn sẽ thu được thông tin dưới đây:

(Mảng)

`rname[]`	Chứa các hàng mã hóa từ A đến Z.
`Status[][]`	Chứa kí tự ' ' hoặc 'R' biểu diễn trạng thái ghế.
`cnum[]`	Chứa số lượng các ghế trên từng hàng. Được dùng trong dòng 6 theo mẫu `cnum[ridx]`.

(Cấu trúc)

`empty.row`	Chứa mã hàng của số lượng các ghế trống liên tiếp nhau được lựa chọn. Dữ liệu dạng ký tự.
`empty.no`	Chứa số thứ tự tính từ phía ngoài cùng bên trái của các ghế trống liên tiếp nhau được lựa chọn. Dữ liệu dạng số nguyên.

(Biến)

`Number`	Số lượng lượng các ghế trống liên tiếp nhau cần tìm; được truyền vào như một tham số.
`Ridx`	Dòng 7 có "`status[ridx][cidx]`", và dòng 10 có "`status[ridx][eidx]`", biến này tương ứng với chỉ số i để chỉ ra hàng ghế trong ma trận ghế.
`Cidx`	Dòng 7 có "`status[ridx][cidx]`", biến này tương ứng với chỉ số j để chỉ ra số ghế trong ma trận ghế.
`Eidx`	Trong các dòng 10 và 22 có "`status[ridx][eidx]`", biến này tương ứng với chỉ số j để chỉ ra số ghế trong ma trận ghế.
`Flg`	Các dòng 19-20 có "`if (flg == 0) return NULL;`", do đó biến `flg` là biến trạng thái để chỉ ra có tìm thấy số lượng các ghế trống liên tiếp được yêu cầu được tìm thấy hay không; giá trị 0 tức là không tìm thấy, và bằng 1 tức là tìm thấy.

(Cấu trúc tiến trình)

Các dòng 5-18	Khối này dùng để tìm các ghế trống liên tiếp nhau. Gồm 3 vòng lặp lồng nhau. Các dòng 5-18: biến `ridx` được dùng để duyệt các hàng từ trên xuống dưới trong quá trình tìm kiếm. Lines 6-16: biến `cidx` được dùng để kiểm tra các ghế trống từ trái sang phải. Lines 9-13: biến `eidx` được dùng để kiểm tra các ghế trống từ phải sang trái.
Các dòng 19-20	Khối này được thực hiện khi không tìm thấy các ghế trống liên tiếp nhau. Có thể dễ dàng nhìn thấy tại dòng 20: "`return NULL;`".
Các dòng 21-25	Khối này được thực hiện khi tìm thấy các ghế trống liên tiếp nhau. Các dòng 21-22: biến `eidx` được dùng để chuyển sang trạng thái ghế “đã có người” từ phải qua trái. dòng 23: Gán mã hàng (kiểu `char`) vào biến `empty.row`. dòng 24: Gán số (kiểu `int`) vào biến `empty.no`. dòng 25: Hàm trả về địa chỉ của biến "`empty`".

Trong quá trình tổ chức chương trình như bảng trên, bạn sẽ dễ dàng nhìn ra

các câu hỏi và những quan sát về vai trò của mảng cũng như các vai trò và các chức năng của các biến.

(1) Vai trò của mảng "rname[]"

[Mổ tả chương trình] không đem lại lời giải thích rõ ràng. Ngay cả ở trong [Chương trình], chúng ta cũng không thể tìm ra bất cứ tiến trình nào sử dụng mảng này. Điều đó gợi ý rằng sẽ có một trong các ô trống chứa mảng này.

(2) Cách sử dụng biến "eidx" và vai trò độc lập với biến "cidx"

Giá trị khởi tạo của biến "eidx" ở dòng 9 là giá trị trong ô trống B, vì vậy ngay tại thời điểm gán, giá trị đó chưa bị xóa. Liên hệ với điều này, chúng ta không cần biết ngay tại sao hai biến chỉ số "cidx" và "eidx" đều cùng được dùng cho các số ghế. Các dòng 21 và 22 bao gồm tiến trình tương tự như mẫu của các dòng 9 và 10, vì vậy nó có thể đưa ra cho chúng ta một đầu mối.

(3) Cách sử dụng biến "flg"

Dòng 3 khởi tạo biến bằng 0. Sau đó, trong dòng 11, nó lại được gán bằng 0, nhưng không có tiến trình nào đặt giá trị của biến này là 1. Do đó, trong một ô trống nào đó giữa các dòng 3 và 11, giá trị của biến phải được gán là 1.

Dưới đây là tóm tắt của những việc được yêu cầu trong từng ô trống:

Ô trống A: Tiến trình được thực hiện đầu tiên
"status[ridx][cidx]" là ‘ ‘ (còn ghế trống)
Ô trống B: Giá trị khởi tạo của biến "eidx"
Ô trống C: Mã hàng tại đó một số các ghế trống liên tiếp nhau được tìm thấy; được gán cho biến "empty.row"
Ô trống D: Số hiệu ghế ngoài cùng bên trái của một số yêu cầu các ghế trống liên tiếp nhau; được gán cho biến "empty.no"

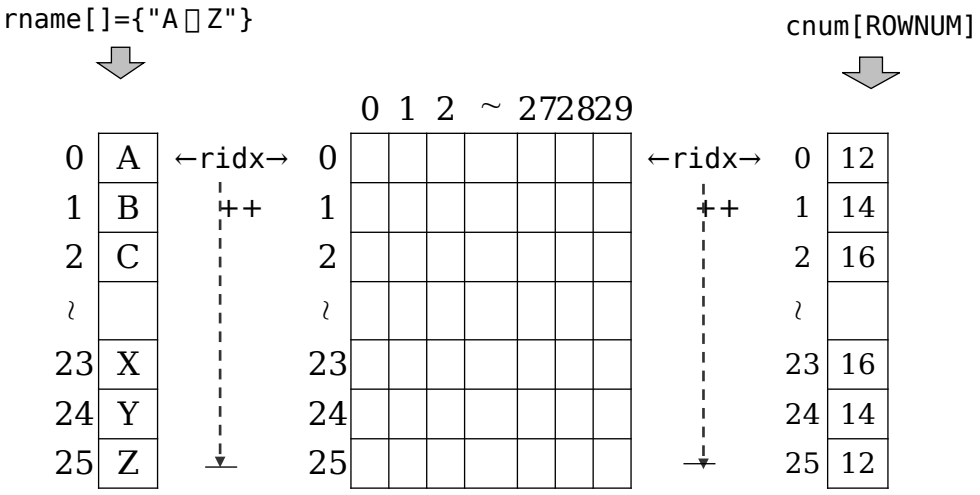
Bạn nên điền vào các ô trống khi bạn hiểu được những gì đã biết và những gì chưa biết với dữ liệu tới thời điểm hiện tại. Tiếp đó, chúng ta cùng điền vào các ô trống khi trả lời các câu hỏi con.

(1) Vai trò của mảng "rname[]" → Ô trống C

Để hiểu vai trò của mảng "rname[]", chúng ta có thể vẽ ra bảng dưới đây, các mảng được phân ra làm 3 có quan hệ lẫn nhau, sử dụng như một lời gợi ý để

status[ROWNUM][MAXNUM]

chỉ ra số lượng các phần tử trên từng hàng và cách sử dụng chúng.



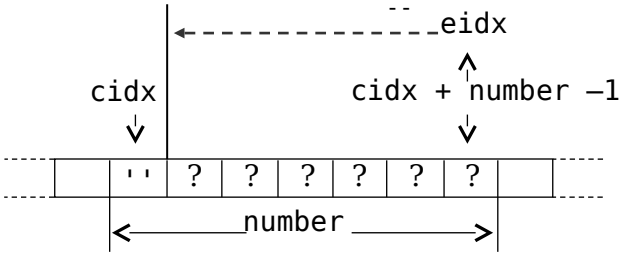
Hình 1

Một đầu mối khác là ở dòng 6 chỉ ra số lượng các ghế trong từng hàng qua cách dùng biến cnum[ridx]. Từ hình 1 và cnum[ridx], chúng ta thấy rằng rname[ridx] biểu diễn mã hàng tương ứng với biến ridx. Nói một cách khác, mảng rname[] có chức năng chuyển đổi biến ridx thành mã hàng tương ứng. Nguyên nhân mà biến ridx cần phải chuyển đổi tương ứng sang mã hàng là trong dòng 23, mã hàng được gán cho biến buy.row. Từ những nhận xét trên, chúng ta thấy ý trả lời đúng cho ô trống C là (d) rname[rdix]. Các mảng như mảng rname[] và mảng cnum[] được gọi là các bảng chuyển đổi. Bạn nên nhớ cách dùng chúng vì đó là một kĩ thuật thường được sử dụng.

(2) Cách sử dụng biến "eidx" và vai trò độc lập với biến "cidx" → Các ô trống B và D

Để hiểu hoạt động của biến "eidx", chúng ta xác nhận giá trị cuối cùng, những lúc tăng và giảm, và cách dùng của biến. Trong dòng 9, biến "eidx" giảm cho đến khi giá trị tiếp theo của nó bằng với giá trị biến "cidx". Trong dòng tiếp theo, dòng 10, chương trình kiểm tra ghế được xác định bởi biến "eidx" đã có người ngồi hay chưa. Nói một cách khác, biến "eidx" hoạt động như một biến chỉ số để kiểm tra một dãy các ghế còn trống hay không, và giá trị khởi tạo của nó nên là giá trị chỉ ra phần tử ngoài cùng bên phải cần được kiểm tra. Bởi vậy, giá trị khởi tạo của biến "eidx" phải bằng tổng của biến "cidx" và số lượng ghế yêu cầu được lưu trong biến "number", trừ đi 1. Do đó, đáp án cho ô trống B là (e) cidx + number − 1.

Hình dưới đây minh họa cho những lập luận trên:



Hình 2

Đáp án chính xác cho ô trống B cũng có thể được tìm ra nhờ tiến trình thay đổi trạng thái của số lượng các ghế tìm được thành “đã đặt chỗ” bắt đầu từ ghế được chỉ định bởi biến "cidx" trong các dòng 21 và 22. Tuy nhiên, hai tiến trình này có khác nhau chút ít trong cách xác định các ghế liên nhau. Trong dòng 9, từ khi biết được ghế được chỉ định trong biến "cidx" là còn trống, biến "eidx" chỉ cần phải duyệt đến ghế ngay sau ghế mà biến

"cidx" trỏ tới. Trong dòng 21, ngược với tiến trình trên, trạng thái của s
lượng các ghế tìm thấy được chuyển thành "đã đặt ch
"eidx" cần được duyệt lùi về giá trị của biến "cidx".

Bây giờ, chúng ta hãy tổng kết vai trò của các biến "cidx" và "eidx".
Chúng ta thấy rằng biến "cidx" có chức năng chỉ ra ghế ngoài cùng bên trái
của các ghế trống liên tiếp nhau, trong khi biến "eidx" có chức năng kiểm tra
số lượng các ghế liên tiếp nhau cần tìm còn hay không. Như vậy, giá trị cần
được gán cho biến "empty.n" trong dòng 24 là số ghế tương ứng với biến
"cidx", đó là chỉ số của ghế ngoài cùng bên trái của các ghế trống liên tiếp
nhau. Để chuyển đổi chỉ số sang số ghế, chúng ta cần cộng thêm 1 vào chỉ
số, nên ô trống D phải là `cidx + 1`. Vậy đáp án đúng là (b).

(3) Cách sử dụng biến "flg" → Ô trống A

Khi chúng ta lần theo hoạt động của biến "flg", chúng ta thấy rằng ban đầu biến được khởi tạo bằng 0 trong dòng 3 và lại được gán bằng 0 trong dòng 11. Do đó, giá trị 1 cần được gán cho biến ở chỗ nào đó giữa các dòng 3 và 11. Hoặc bạn có thể phát hiện ra ở tiến trình trong dòng 14, giá trị của biến được so sánh với 1, và kết luận rằng giá trị 1 cần được gán cho biến giữa các dòng 3 và 14. Dù bằng cách lập luận nào, việc gán giá trị cho biến phải được điền vào ô trống A. Bằng những lời giải thích của các dòng trên, chúng ta biết rằng giá trị 1 nên được gán ngay sau khi tìm ra ghế ngoài cùng bên trái của dãy ghế trống yêu cầu, nên chúng ta có thể xác nhận đấy là vị trí thích hợp. Vì thế, (f) `flg = 1` phải được điền vào ô trống A.

Bây giờ tất cả các ô trồng đều đã được điền, chúng ta hãy kiểm tra chương trình hoạt động ra sao khi nó tìm thấy số lượng các ghế trống liên tiếp nhau được yêu cầu. Ở đây chúng ta bỏ qua trường hợp bài toàn. Để nghiệm thuận tiện hơn, chúng ta thêm vào các dòng phụ sử dụng giả ngôn ngữ cho chương trình ở trên vì thế chương trình có thể dễ dàng được kiểm tra.

Khi một ghế trống được tìm thấy trong dòng 7, ở ô trống A của dòng 8, giá trị 1 được gán cho biến `flg`. Vòng lặp của các dòng 9~13 kiểm tra ở đó có đủ số lượng ghế trống yêu cầu không. Nếu đủ, biến `flg` giữ nguyên giá trị 1, và chương trình đến dòng 14 để xử lý. Điều kiện của dòng 14 là đúng, chương trình thoát khỏi vòng lặp của các dòng 6~16 và nhảy đến dòng 17. Điều kiện của dòng 14 cũng đúng, chương trình thoát khỏi vòng lặp của các dòng 5~18 và nhảy đến dòng 19. Điều kiện của dòng 19 là sai, nên dòng 20 không được thực hiện, và chương trình chuyển đến tiến trình đặt ghế trước trong dòng 21 và 22.

Như vậy, chúng ta chắc chắn rằng chương trình hoạt động chính xác khi `flg = 1` được điền vào ô trống A.

Câu hỏi được phân loại ở mức cơ bản, với độ khó thấp. Nếu bạn gặp khó khăn với việc trả lời câu hỏi này, thì hãy sử dụng các gợi ý ở trên để tìm cách đọc các chương trình và phương pháp trả lời các câu hỏi cho riêng bạn. Nếu bạn trả lời được trọn vẹn câu hỏi này, thì hãy dùng nó như một bài tập để trả lời các câu hỏi với mức độ khó cao hơn; tìm một vài cách để dẫn dắt bạn đến những trả lời đúng, và tập hợp chúng lại.

Câu hỏi 2 — Lập trình C

Q2. Đọc mô tả chương trình và chương trình C dưới đây rồi trả lời các Câu hỏi con 1 và 2.

[Mô tả chương trình 1]

Đây là chương trình đọc một chương trình nguồn được viết bằng ngôn ngữ lập trình C từ thiết bị vào chuẩn, loại bỏ các chú thích, sau đó đưa kết quả r thiết bị ra chuẩn.

(1) Mô tả kí hiệu của các chương trình nguồn:

(i) "Các giải thích" được xử lý bởi chương trình là các xâu kí tự bắt đầu bằng "/*" và kết thúc bằng "*/", ngoài ra chúng còn bao gồm các hằng kí tự, các xâu kí tự và các chú thích.

(ii) Các kí tự hợp lệ được cho trong bảng.

Space	0	@	P	`	p
!	1	A	Q	a	q
"	2	B	R	b	r
#	3	C	S	c	s
$	4	D	T	d	t
%	5	E	U	e	u
&	6	F	V	f	v
'	7	G	W	g	w
(	8	H	X	h	x
)	9	I	Y	i	y
*	:	J	Z	j	z
+	;	K	[	k	{
,	<	L	\	l	|
-	=	M	]	m	}
.	>	N	^	n	~
/	?	O	_	o	

(iii) Các mô tả được cho dưới đây là không hợp lệ.

- Các giải thích lồng nhau

  Ví dụ `/* aaaaa /* bbbbbb */ ccccc */`

- Kí hiệu có 3 kí tự dành cho các kí tự đồ họa

  `??= ??( ??' ??< ??> ??) ??! ??-`

(iv) Không chứa các lỗi ngữ pháp.

(2) Chương trình 1 loại bỏ các chú thích theo đúng như thủ tục bên dưới. Từ đó chương trình dễ dàng xử lí việc phân tích các hằng kí tự, các xâu kí tự và các chú thích, chúng có thể không được nhận dạng chính xác thuộc vào việc viết mã nguồn chương trình, đó là nguyên nhân gây lỗi.

(i) Khi tìm ra một dấu nháy đơn hay một dấu nháy kép, chương trình hiểu nó như là nơi bắt đầu của hằng kí tự hay xâu kí tự và sau đó sử dụng hàm quote để đọc và xuất ra xâu kí tự cho đến khi chương trình phát hiện ra dấu nháy đơn hay một dấu nháy kép kết thúc tương ứng.

(ii) Khi phát hiện ra xâu "/*", chương trình hiểu đó là bắt đầu đoạn chú thích và bỏ đi các kí tự trước xâu "*/" xuất hiện đầu tiên.

(3) Dưới đây là một ví dụ loại bỏ chú thích của chương trình 1.

Mã nguồn đọc dữ liệu

```
/* Chương trình dùng hàm fgets để xuất
 * một dòng từ file lên màn hình. */
#include <stdio.h>
int main( void )
{
   FILE *stream;    /* con trỏ file */
   char line[100];  /* dữ liệu đầu vào */

   if( (stream = fopen( "crt_fgets.txt", "r" )) != NULL )
   {
      if( fgets( line, 100, stream ) == NULL)
         printf( "fgets error\n" );  /* thông báo lỗi */
      else
         printf( "%s", line);
      fclose( stream );
   }
}
```

Ghi kết quả sau khi loại bỏ các chú thích

```
#include <stdio.h>
int main( void )
{
   FILE *stream;
   char line[100];

   if( (stream = fopen( "crt_fgets.txt", "r" )) != NULL )
   {
      if( fgets( line, 100, stream ) == NULL)
         printf( "fgets error\n" );
      else
         printf( "%s", line);
      fclose( stream );
   }
}
```

Hình. Ví dụ thực hiện loại bỏ chú thích

[Chương trình 1]

```
#include <stdio.h>
void quote( char );

main()
{
    int c1, c2;

    while ( (c1 = getchar()) != EOF ) {
        /* phát hiện dấu nháy đơn    */
        if ( c1 == '\'' ) quote( '\'' );
        /* phát hiện dấu nháy kép    */
        else if ( c1 == '\"' ) quote( '\"' );
        /* phát hiện dấu '/'       */
        else if ( c1 == '/' ) {
            c2 = getchar();
            /* khi kí tự tiếp theo là dấu sao */
            if ( c2 == '*' ) {
                /* loại bỏ xâu kí tự chú thích */
                while ( 1 ) {
                    while ( (c1 = getchar()) != '*' );
                    c2 = getchar();
                    if ( c2 == '/' ) break;
                }
            }
            /* trường hợp còn lại    */
            else {
                putchar(c1);
                putchar(c2);
            }
        }
        else putchar(c1); /* in ra một kí tự */
    }
}

void quote( char c )
{   /* lấy ra từng hằng kí tự và xâu kí tự    */
```

```
    char cc;

    putchar(c);
    while ( (cc = getchar()) != c ) putchar(cc);
    putchar(cc);
}
```

Câu hỏi con 1

Từ nhóm câu trả lời bên dưới, chọn đoạn mã làm sai lệch hoạt động khi được đem vào chương trình 1.

Nhón đáp án:

a) `/*   "aaaaaaa"  */`
b) `/* aaa  'a'     */`
c) `if ( c == '\'' ) {`
d) `printf( "  \'     " );`
e) `printf( "aaa  /*  comment  */ \n" );`

[Mô tả chương trình 2]

Để giải quyết vấn đề được chỉ ra trong phần (2) của **[Mô tả chương trình 1]**, chương trình 2 được viết lại dưới đây.

(1) Tiến trình được chia thành ba chế độ: hằng kí tự, xâu kí tự và chú thích.
(2) Sự xuất hiện của dấu nháy đơn báo hiệu bắt đầu hoặc kết th
"chế độ hằng kí tự". Tuy nhiên, nó không được áp dụng cho đoạn mã được mô tả mở rộng bởi việc sử dụng kí tự \"", cho đoạn mã ở trong xâu kí tự, hoặc đoạn mã ở trong phần chú thích.
(3) Sự xuất hiện của dấu nháy kép báo hiệu bắt đầu hoặc kết th
"chế độ xâu kí tự". Tuy nhiên, nó không được áp dụng cho đoạn mã được mô tả mở rộng bởi việc sử dụng kí tự "\", cho đoạn mã ở trong hằng kí tự, hoặc đoạn mã ở trong phần chú thích.
(4) Sự xuất hiện của cặp xâu "/*" và "*/" báo hiệu bắt đầu và kết thúc của "chế độ chú thích". Tuy nhiên, nó không được áp dụng cho đoạn mã ở trong hằng kí tự hoặc cho đoạn mã ở trong xâu kí tự.

[Chương trình 2]

```
#include <stdio.h>
main()
{
    int c1, c2;
    int c_mode = 0;  /* khởi tạo kết thúc chế độ chú thích     */
    int quote1 = 0;  /* khởi tạo kết thúc chế độ hằng kí tự    */
    int quote2 = 0;  /* khởi tạo kết thúc chế độ xâu kí tự    */

    for ( c1 = getchar(); ( c2 = getchar()) != EOF; c1 = c2 ) {

        if ( !c_mode ) {     /* khi hết chế độ chú thích */
       /* phát hiện nếu kí tự ở trong một hằng kí tự hoặc trong một xâu kí
tự.*/
            if ( [A] && c1 == '\\' ) {
                putchar(c1);
                putchar(c2);
                c2 = getchar();
                continue;
            }
            /* kiểm tra nếu dấu nháy đơn không nằm trong xâu kí tự */
            else if ( !quote2 && c1 == '\'' )
                [B];
            /* kiểm tra nếu dấu nháy kép không nằm trong hằng kí tự */
            else if ( !quote1 && c1 == '\"' )
                [C];
            /* kiểm tra nếu kí tự / và * không nằm tron hằng kí tự */
          /* và không nằm trong xâu kí tự */
            else if ( [D] && c1 == '/' && c2 == '*' ) {
                [E];
                c2 = getchar();
                continue;
            }
            putchar(c1);
        }
```

```
        else {
            if ( c1 == '*' && c2 == '/' ) {  /* kết thúc câu chú thích? */
                E;
                c2 = getchar();
            }
        }
    }
    putchar(c1);
}
```

Câu hỏi con 2

Từ các nhóm câu trả lời dưới đây, chọn các câu trả lời đúng để điền vào trong ô trống [ ] trong chương trình 2.

Nhóm câu trả lời cho A và D:

a) `!quote1`
b) `!quote2`
c) `(!quote1 || !quote2)`
d) `(!quote1 && !quote2)`
e) `(quote1 || quote2)`
f) `(quote1 && quote2)`

Nhóm câu trả lời cho B, C và E:

a) `c_mode = !c_mode`
b) `c_mode = quote1 && quote2`
c) `quote1 = !quote1`
d) `quote1 = !quote2`
e) `quote1 = quote2`
f) `quote2 = !quote1`
g) `quote2 = !quote2`
h) `quote2 = quote1`

Đáp án câu 2

Loại bỏ các chú thích trong mã nguồn của ngôn ngữ C

Đáp án đúng

[Câu hỏi con 1] c

[Câu hỏi con 2] A – e, B – c, C – g, D – d, E – a

Giải thích

Đây là câu hỏi liên quan đến chương trình đọc mã nguồn được viết bằng ngôn ngữ C, loại bỏ các chú thích, và đưa ra kết quả. Nội dung của câu hỏi chiếm số lượng trang khá lớn, nhưng bạn có thể trả lời được câu hỏi này trong thời gian ngắn bằng việc sử dụng các mô tả chương trình và các chú thích ở trong chương trình, vì thế hãy cố gắng trả lời các câu hỏi này mà không lo ngại về độ dài của chúng.

[Câu hỏi con 1]

Đây là câu hỏi tìm trường hợp gây ra lỗi khi đem chương trình ra thực hiện. Không cần đọc [Chương trình 1], từ thủ tục được giải thích trong phần (2) của [Mô tả chương trình], bạn có thể chỉ ra trường hợp xảy ra lỗi. Cụ thể hơn, chúng ta hãy đọc phần (2) [Mô tả chương trình]: "... chúng có thể không được nhận dạng chính xác ... đó là nguyên nhân gây lỗi". Chúng ta có thể tìm ra đáp án dựa trên chú thích đó, vì việc này được làm theo chú thích của thủ tục loại bỏ các chú thích. Để kiểm tra chương trình hoạt động ra sao với từng lựa chọn trong nhóm trả lời, chúng ta hãy xem dưới đây:

(a) và (b) rơi vào trường hợp (ii), xóa mọi kí tự xâu "/*" đến xâu "*/" được hiểu là các chú thích. Thao tác này không phát sinh bất kì lỗi nào.

(c) rơi vào trường hợp (i) xuất ra mọi kí tự giữa dấu nháy đơn (') và dấu nháy đơn thứ hai được hiểu là hằng kí tự. Sau đó, dấu nháy đơn thứ ba được hiểu là bắt đầu của hằng kí tự khác, kí tự sẽ được đưa ra nguyên vẹn. Tuy nhiên, trong thực tế, dấu nháy đơn thứ ba chỉ ra vị trí kết thúc của hằng kí tự, vì thế nó không được nhận dạng chính xác. Do đó, khi xâu "/*" xuất hiện trong dòng tiếp theo chỉ ra các lời chú thích thì cũng được đưa ra mà không cần kiểm tra, đây là nguyên nhân sinh ra lỗi. Một ví dụ đặc trưng được chỉ ra dưới

đây, các chú thích được xuất ra.

```
if ( c == '\'' ) {
/* Is "c" a single quotation mark?  */
```

(d) và (e) rơi vào trường hợp (i), xuất ra mọi kí tự giữa dấu nháy kép đầu tiên (") và dấu nháy kép thứ hai được hiểu là xâu kí tự. Thao tác này không phát sinh bất kì lỗi nào.

Vì thế, đáp án đúng là (c).

Giờ đây, chúng ta đã trả lời xong câu hỏi mà không thực sự cần đọc chương trình; đáp án thu được chỉ dựa vào phần mô tả chương trình. Trong bài thi thực tế, câu hỏi của dạng này nên được trả lời theo cách trên, và sau đó, nếu còn thừa thời gian, bạn có thể đọc kĩ chương trình để kiểm tra lại đáp án.

[Câu hỏi con 2]

Các nhóm câu trả lời được chia thành các câu cho các ô trống A và D và các câu cho các ô trống B, C, và E. Có thể có một vài đọc giả băn khoăn khi nhìn thấy nhóm trả lời A và D. Mỗi một ô trống đó ở trong một nhánh lệnh "if". Nh chung, điều kiện thường được biểu diễn bởi việc liên kết nội dung biến với giá trị bởi việc sử dụng phép toán quan hệ, ví dụ `ch != '\\'`. Tuy nhiên, không có lựa chon nào trong nhóm trả lời của các ô trống A và D có dạng đó. Tất cả chúng chỉ có các biến hoặc dấu chấm than (`!`) đặt trước các biến, không kèm theo đó các phép toán quan hệ.

Các mô tả điều kiện trong dạng trên là cơ bản về mặt ngữ pháp, nhưn phần lớn thí sinh có thể chưa từng gặp trước đó bởi vì việc dùng kí hiệu này đôi khi bị cấm bởi các luật viết mã tại các môi trường phát triển, hoặc vì ý nghĩa ban đầu có thể được viết bằng các phép toán quan hệ khác. Trong ngôn ngữ C, các điều kiện `!= 0` và `== 0` có thể bị bỏ qua trong mô tả. Ví dụ, nếu một biểu thức điều kiện là `quote1 != 0`, nó có thể được viết là `quote1`, bỏ qua phép toán quan hệ và bất cứ thứ gì sau đấy. Nếu một biểu thức điều kiện là `quote1 == 0`, nó tương đương với cách viết `!quote1`. Đó là vì các dạng dữ liệu trong C không chứa kiểu lô-gíc; thay vào đó, kiểu số nguyên (`int`) được sử dụng, giá trị 0 là sai và mọi giá trị khác 0 có nghĩa là đúng. Vì thế, một phép toán quan hệ trả về giá trị khác 0 nếu biểu thức đã cho là đúng và trả về 0 nếu nó là sai.

Bây giờ, dù là bạn không biết vấn đề gì đã được trình bày ở trên, bạn vẫn có thể hình dung ra như sau. Để ý lệnh "if" bên trên hai dòng ô trống A với chú thích "khi hết chế độ chú thích". Điều kiện của nó được chỉ ra là "`!c_mode`". Tại đó, người đọc nên hiểu rằng `!c_mode` nghĩa là `c_mode == 0` (bởi vì giá trị khởi tạo của biến c_mode là kết thúc chế độ, và giá trị của nó là 0).

Trong phần giải thích, đầu tiên chúng ta sẽ thảo luận về các ô trống A và D và sau đó giải quyết các ô trống B, C, và E. Các tiến trình trong các ô trống của cả hai nhóm được giải thích trong các câu chú thích, vì thế chúng ta chỉ cần chuyển chúng thành các biểu thức trong chương trình. Nói theo kĩ thuật, bất kì

số khác 0 nào cũng có thể được gán là đúng, nhưng trong các chú thích ở đây, chúng ta sẽ dùng giá trị "1".

Ô trống A

Dựa vào các chú thích, câu lệnh "if" chứa ô trống A đang kiểm tra nếu kí tự "\" nằm trong một hằng kí tự hay một xâu kí tự. Ở đây, "việc kiểm tra kí tự \" tương ứng với biểu thức "`c1 == '\\'`", vì thế ô trống A sẽ tương ứng với "trong một hằng kí tự hoặc trong một xâu kí tự". Chúng ta cần phải chuyển sang một biểu thức trong ngôn ngữ C. "trong ..." gợi ý rằng chế độ hằng kí tự hoặc chế độ xâu kí tự sẽ được bắt đầu (1). Việc chuyển đổi đó theo cách dễ hiểu, chúng ta có (`quote1 != 0 || quote2 != 0`), nhưng nó không có trong nhóm câu trả lời. Tuy vậy, khi xem lại nhóm câu trả lời một lần nữa, chúng ta thấy biểu thức `quote1 || quote2` dựa trên cơ sở là chế độ bắt đầu (1) và chế độ kết thúc (0) của `quote1` và `quote2` tương ứng với giá trị đúng (1) và sai (0). Vì vậy, đáp án đúng cho ô trống A là (e).

Ô trống D

Dựa vào các chú thích, câu lệnh "if" chứa ô trống D đang kiểm tra nếu "kí tự / và * không nằm trong hằng kí tự và xâu kí tự". Ở đây, việc kiểm tra "/" và "*" tương ứng với biểu thức `c1 == '/' && c2 == '*'`, vì thế ô trống D sẽ tương đương với "không nằm trong hằng kí tự và xâu kí tự". Chúng ta cần phải chuyển sang một biểu thức trong ngôn ngữ C. "Không..." hiểu là điều kiện của chế độ hằng kí tự là kết thúc (0). Do đó, biểu thức "`!quote1`" là cần thiết. Tương tự như vậy, đó là điều kiện của chế độ xâu kí tự là kết thúc (0), vì thế biểu thức phủ định `!quote2` cũng là cần thiết. Điều kiện "and" hiểu là chúng ta cần liên kết biểu thức `!quote1` và biểu thức `!quote2` bằng phép toán &&. "Vì vậy, đáp án đúng là `!quote1 && !quote2`), đó là ý (d).

Ô trống B

Dựa vào các chú thích, ô trống B là tiến trình được thực hiện khi "chế độ chú thích kết thúc" và khi không có "kí tự\ được phát hiện trong hằng kí tự hoặc xâu kí tự" và khi "dấu nháy đơn được tìm thấy nhưng không nằm trong một xâu kí tự". Dấu nháy đơn (') được dùng để chỉ ra chỗ bắt đầu và kết thúc của một hằng kí tự, vì thế việc cần phải làm là chuyển chế độ hằng kí tự. Cụ thể hơn, nếu chế độ đó là bắt đầu (1), nó cần được chuyển sang chế độ kết thúc (0); và ngược lại. Nói cách khác, giá trị phủ định của biến `quote1` được gán cho biến `quote1`, do đó chúng ta cần biểu thức "`quote1 = !quote1`". Đáp án đúng cho ý B là (c).

Nếu bạn không tin tưởng vào kết luận rằng đó là điều kiện để bắt đầu hoặc kết thúc một hằng kí tự, lưu ý đến điều kiện đối lập, ví dụ, vị trí nơi dấu nháy đơn được tìm thấy nhưng không chỉ ra nơi bắt đầu hoặc kết thúc của một hằng kí tự. Điều đó xảy ra nếu "chế độ chú thích bắt đầu" hoặc ngay sau "kí tự\ được tìm ra bên trong hằng xâu kí tự hoặc xâu kí tự" hoặc nếu một dấu nháy đơn được tìm thấy "bên trong một xâu kí tự". Kiểm tra điều kiện phủ định, và bạn có thể kiểm tra điều kiện bắt đầu hoặc kết thúc một hằng kí tự.

Ô trống C

Ô trống C có cùng cách suy luận như ô trống B.

Dựa vào các chú thích, tiến trình nên được điền vào ô trống C là khi "chế độ chú thích kết thúc" và khi không có "kí tự\ được tìm thấy trong một hằng xâu kí tự hoặc trong một xâu kí tự" và khi "dấu nháy kép được tìm ra nhưng không nằm trong hằng kí tự". Dấu nháy kép (") được dùng để chỉ ra vị trí bắt đầu hoặc kết thúc của một xâu kí tự, do đó việc cần thiết phải được làm ở đây

là chuyển chế độ xâu kí tự. Vì vậy, biểu thức `quote2 = !quote2` được điền vào ô trống C, đáp án là (g).

Ô trống E

Ô trống E xuất hiện ở 2 vị trí. Một là khi "chế độ chú thích kết thúc" và khi "kí tự / và * không được tìm thấy bên trong một hằng kí tự và một xâu kí tự". "Kí tự / và *" (tức là xâu/"*") chỉ ra vị trí bắt đầu của một chú thích, do đó, việc cần phải làm ở đây là chuyển chế độ chú thích từ kết thúc (0) sang bắt đầu (1). Tại vị trí còn lại là khi không phải "chế độ chú thích kết thúc" (hiểu là khi đó chế độ là đay ở chú thích) và tại nơi "kết thúc chú thích". Do đó, điều kiện của câu lệnh "`if`" bị đảo ngược, tức là, để tìm ra "* và /" hay "`*/`", nơi kết thúc của chú thích. Việc cần phải làm ở đây là chuyển chế độ chú thích từ bắt đầu (1) sang kết thúc (0). Đoạn mã có thể làm cả hai việc này là "`c_mode = ! c_mode`". Vì vậy, đáp án cho ô trống E là (a).

(Các chú ý thêm)

Cuối cùng, mặc dù không liên quan trực tiếp đến các câu hỏi, chúng ta sẽ thêm một lưu ý ngắn gọn ở đây về kí hiệu có 3 kí tự dành cho các kí tự đồ họa xuất hiện trong phần (iii) của [Mô tả chương trình 1].

Kí hiệu có 3 kí tự dành cho các kí tự đồ họa còn được gọi là ba kí tự đồ họa liên tiếp, và nó liên quan đến tâp hợp các xâu kí tự để biểu diễn những kí tự như "#" và "[" trong môi trường mà ở đó các kí tự này không thể sử dụng được. Ví dụ, "??=" biểu diễn "#", "??(" biểu diễn "[" và “??’” biểu diễn “^”.

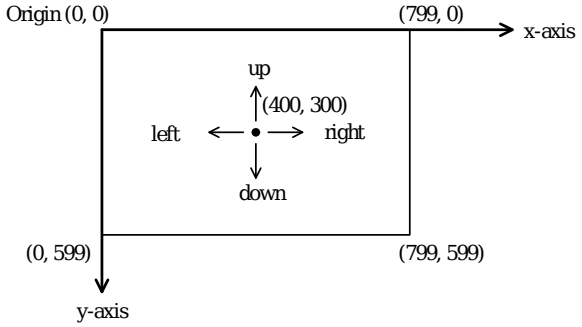
Câu hỏi 3 Lập trình C

Q3. Đọc mô tả chương trình và chương trình C dưới đây rồi trả lời các Câu hỏ
con.

[Mô tả chương trình]

Hàm execute thực hiện vẽ các đường thẳng sử dụng con trỏ trên màn hình.

(1) Một màn hình bitmap với kích thước chiều rộng 800 pixels và chiều cao 600 pixels. Hình 1 biểu diễn hệ tọa độ của màn hình. Con trỏ (* trong Hình 1) có tọa độ và hướng được thể hiện trên màn hình. Con trỏ có khả năng di chuyển theo 4 hướng: lên trên, xuống dưới, sang trái, sang phải. Con trỏ sẽ được vẽ khi nó di chuyển.



Hình 1 Hệ tọa độ của màn hình và trạng thái ban đầu của con trỏ

(2) Con trỏ được biểu diễn bằng một biến mark có kiểu là bản ghi MARKER. Khi chương trình bắt đầu, vị trí của con trỏ được gán giá trị (400, 300) và di chuyển hướng lên trên.

```
typedef struct {
    int x;   /* tọa độ x của con trỏ */
    int y;   /* tọa độ y của con trỏ */
    int dir; /* hướng di chuyển   0:phải, 1:lên, 2:trái, 3:xuống */
} MARKER;
MARKER mark = {400,  /* tọa độ x ban đầu */
               300,  /* tọa độ y ban đầu */
               1     /* hướng di chuyển ban đầu (lên) */
               };
```

(3) Con trỏ được điều khiển thông qua các lệnh. Mỗi lệnh được biểu diễ
bằng một bản ghi kiểu INST bao gồm mã lệnh và giá trị.

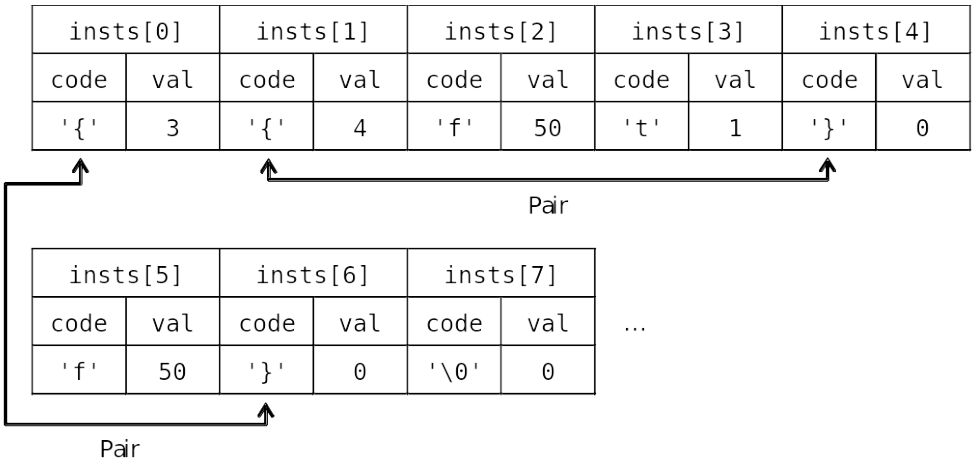
```
typedef struct {
    char code;   /* mã lệnh */
    int  val;    /* giá trị */
} INST;
```

Các lệnh được lưu trữ trong mảng `insts` (mảng các bản ghi kiểu `INST`) và được sắp xếp lần lượt theo thứ tự thực hiện.

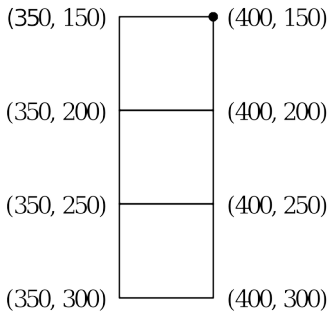
(4) Bảng sau liệt kê tất cả các mã lệnh và mô tả tương ứng.

Mã Lệnh	Mô tả
`{`	Lệnh lặp, lặp `val` lần các lệnh bắt đầu từ lệnh ngay sau nó tới lệnh cuối cùng trước lệnh có mã '}' (tạo thành một cặp '{}'). `val` là một số nguyên có giá trị lớn hơn 1.
`t`	Lệnh chuyển hướng, nhiệm vụ thay đổi hướng di chuyển của con trỏ một góc 90° × `val` theo hướng ngược chiều kim đồng hồ. `val` là một số nguyên dương.
`f`	Lệnh di chuyển, di chuyển con trỏ theo hướng hiện tại `val` pixels và vẽ một đường thẳng từ vị trí ban đầu tới vị trí đích. `val` là một số nguyên.
`}`	Đánh dấu kết thúc một đoạn lệnh lặp. `val` không sử dụng.
`\0`	Đánh dấu kết thúc việc thực hiện điều khiển con trỏ. `val` không sử dụng.

(5) Hình 3 biểu diễn kết quả của việc thực hiện hàm `execute` với các lệnh được lưu trong mảng `insts` trong Hình 2. Chú ý, các giá trị tọa độ trong Hình 3 được thêm vào với mục đích giải thích chứ không được thực sự ghi ra trong kết quả.



Hình 2. Ví dụ các lệnh được lưu trữ trong mảng `insts`



Hình 3. Kết quả thực hiện các lệnh trong Hình 2

(6) Hàm sau được sử dụng để vẽ đường thẳng:

```
void drawLine( int x1, int y1, int x2, int y2 );
```

Chức năng: vẽ một đường thẳng nối tọa độ (x1, y1) và tọa độ (x2, y2) trên màn hình.

(7) Các hàm sau thực hiện các chức năng liên quan tới con trỏ:

```
void eraseMarker( MARKER mark );
```

Chức năng: không thể hiện con trỏ trên màn hình.

```
void paintMarker( MARKER mark );
```

Chức năng: thể hiện con trỏ trên màn hình.

(8) Ngay khi con trỏ không được biểu diễn trên màn hình, tọa độ và hướng di chuyển của nó vẫn không thay đổi.

[Chương trình]

```
#define INSTSIZE 100 /* số lệnh tối đa */
#define STACKSIZE 50 /* số vòng lặp tối đa lồng nhau */

typedef struct {
    int x;      /* tọa độ x của con trỏ */
    int y;      /* tọa độ y của con trỏ */
    int dir;    /* hướng di chuyển   0:phải, 1:lên, 2:trái, 3:xuống */
} MARKER;

typedef struct {
    char code;  /* mã lệnh */
    int  val;   /* giá trị */
```

```
} INST;

typedef struct {
    int opno;    /* phần tử thứ No. của mảng insts bắt đầu vòng lặp */
    int rest;    /* số vòng lặp còn lại */
} STACK;
void drawLine( int, int, int, int );
void eraseMarker( MARKER );
void paintMarker( MARKER );

INST insts[INSTSIZE];    /* mảng chứa các lệnh */
MARKER mark = {400,      /* tọa độ x ban đầu */
               300,      /* tọa độ y ban đầu */
               1         /* hướng di chuyển ban đầu (lên) */
              };

void execute(){

    STACK stack[STACKSIZE];
    int opno = 0;    /* phần tử thứ No. của mảnginsts chứa lệnh được thực
hiện */
    int spt = -1;    /* con trỏ stack */
    int dx, dy;

    paintMarker( mark );

    while( insts[opno].code != '\0' ){
        switch( insts[stack].code ){
            case '{':
                    stack[ A ].opno = opno;
                    stack[spt].rest = insts[opno].val;
                    break;
            case 't':
                    mark.dir = B ;
                    break;
            case 'f':
                    eraseMarker( mark );
```

```
                dx = ( mark.dir % 2 == 0 ) ? [C];
                dy = ( mark.dir % 2 == 0 ) ? [D];
                drawLine( mark.x, mark.y,
                            mark.x + dx, mark.y + dy );
                mark.x += dx;
                mark.y += dy;
                paintMarker( mark );
                break;
            case '}':
                if ( stack[spt].rest [E] ){
                    opno = stack[spt].opno;
                    stack[spt].rest--;
                } else {
                    [F];
                }
                break;
        }
        [G];
    }
}
```

Câu hỏi con

Từ các nhóm câu trả lời dưới đây, chọn các đáp án đúng để điền vào trong ô trống ☐ trong chương trình trên.

Nhóm câu trả lời cho A:

a) `++spt` b) `--spt` c) `spt`
d) `spt++` e) `spt--`

Nhóm câu trả lời cho B:

a) `( mark.dir + insts[opno].val ) % 2`
b) `( mark.dir + insts[opno].val ) % 3`
c) `( mark.dir + insts[opno].val ) % 4`
d) `mark.dir + insts[opno].val`
e) `mark.dir + insts[opno].val % 2`
f) `mark.dir + insts[opno].val % 3`
g) `mark.dir + insts[opno].val % 4`

Nhóm câu trả lời cho C và D:

a) `( 1 - mark.dir ) * insts[opno].val : 0`
b) `( 2 - mark.dir ) * insts[opno].val : 0`
c) `( mark.dir - 1 ) * insts[opno].val : 0`
d) `( mark.dir - 2 ) * insts[opno].val : 0`
e) `0 : ( 1 - mark.dir ) * insts[opno].val`
f) `0 : ( 2 - mark.dir ) * insts[opno].val`
g) `0 : ( mark.dir - 1 ) * insts[opno].val`
h) `0 : ( mark.dir - 2 ) * insts[opno].val`
i) `0 : mark.dir * insts[opno].val`
j) `mark.dir * insts[opno].val : 0`

Nhóm câu trả lời cho E:

a) `< 0` b) `< 1` c) `== 0`
d) `> 0` e) `> 1`

Nhóm câu trả lời cho F và G:

a) `mark.dir++` b) `mark.dir--` c) `opno++`
d) `opno--` e) `spt++` f) `spt--`

Đáp án câu 3

Điều khiển con trỏ vẽ màn hình

Đáp án đúng

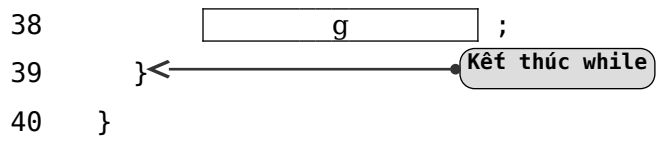
A – a, B – c, C – a, D – h, E – e, F – f, G – c

Giải thích

Đây là câu hỏi liên quan tới một chương trình điều khiển con trỏ vẽ trên màn hình theo các lệnh được lưu trữ trong một mảng. Cấu trúc của chương trình đơn giản và dễ hiểu. Các hàm và các biến được sử dụng, ngoại trừ dx và dy, đều đã được giải thích rõ ràng.

Trong chương trình này, vai trò của cấu trúc dữ liệu và các biến được sử dụng khá phức tạp (cấu trúc mảng được sử dụng). Các chương trình dạng này nên được xem xét các vấn đề lần lượt như sau: vai trò của mảng cấu trúc và các biến → tổ chức cấu trúc của chương trình → hiểu nội dung của chương trình. Thêm vào đó, để hiểu nội dung của chương trình, có thể sử dụng dữ liệu mẫu trong phần câu hỏi. Để giải thích cụ thể hơn, phía dưới là đoạn chương trình của hàm "execute()" cùng với các chú thích thêm gồm số thứ tự của dòng, các khối lệnh và ghi chú.

```
1   void execute(){
2
3     STACK stack[STACKSIZE];
4     int opno □ 0; /* phần tử thứ No. của mảng insts chứa lệnh được thực hiện
        *//
5     int spt □ -1; /* con trỏ stack */
6     int dx, dy;
7
8     paintMarker( mark );
9
10    while( insts[opno].code != '\0' ) {
11        switch( insts[opno].code ) {
12            case '{':
13                [1]  stack[ ____a____ ].opno = opno;
14                     stack[spt].rest = insts[opno].val;
15                     break;
16            case 't':
17                [2]  mark.dir = ____b____;
18                     break;
19            case 'f':
20                     eraseMarker( mark ) ;
21                     dx = ( mark.dir % 2 == 0 ) ? ____c____;
22                     dy = ( mark.dir % 2 == 0 ) ? ____d____;
23                [3]  drawLine( mark.x, mark.y,
24                            mark.x + dx, mark.y + dy );
25                     mark.x += dx;
26                     mark.y += dy;
27                     paintMarker( mark );
28                     break;
29            case '}':
30                     if ( stack[spt].rest ____e____ ){
31                [4]  opno = stack[spt].opno;
32                     stack[spt].rest--;
33                     } else {
34                            ____f____;
35                     }
36                     break;
37        }<-------------------- Kết thúc switch
```



Hàm "execute()"

(Tóm tắt các mảng cấu trúc và biến được sử dụng)

Khi đọc từ đầu chương trình tới dòng 6, từ nội dung của câu hỏi và các ch
thích, vai trò và nội dung của mảng cấu trúc và các biến được làm rõ.

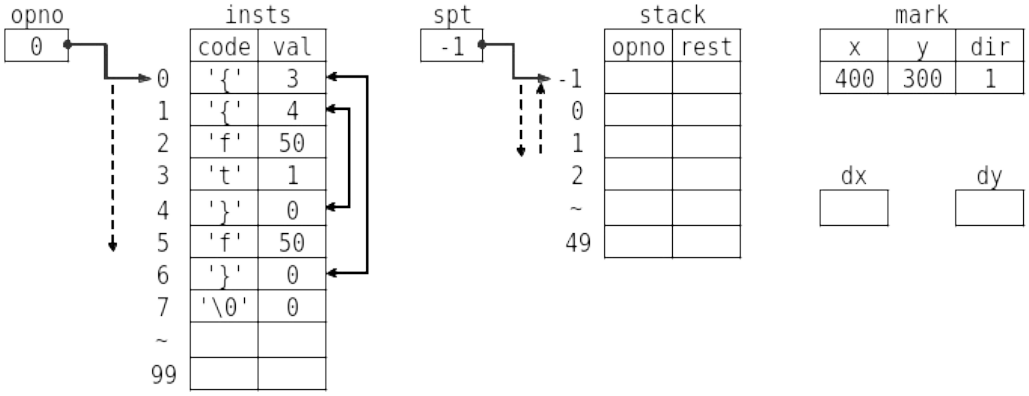
Biến "opno" : chỉ số của mảng "insts", mảng chứa toàn bộ các lệnh dành cho con trỏ. Dòng 4 khởi tạo giá trị 0 cho biến này. Đây là một biến đóng vai trò là chỉ số dùng khi duyệt mảng, do vậy trong chương trình biến này được tăng dần để duyệt và thực hiện các lệnh trong mảng "insts" theo đúng thứ tự.

Biến "spt" : chỉ số của mảng "stack"; dòng 5 khởi tạo giá trị -1 cho biến này. Cho tới thời điểm này, mục đích sử dụng cụ thể của mảng "stack" chưa thực sự rõ ràng, nhưng dựa vào chú thích tại dòng 3 nơi mảng được khai báo, có thể dự đoán rằng mảng này được sử dụng để điều khiển số lượng các vòng lặp lồng nhau. Thêm nữa, vì biến "spt" được khởi tạo giá trị -1, vì vậy biến "spt" có thể sẽ được tăng lên trước khi quá trình lưu trữ vào "stack" được thực hiện. Cuối cùng, vì là một ngăn xếp nên có thể dự đoán rằng biến này sẽ có thể được giảm đi tương ứng khi thực hiện quá trình lấy dữ liệu ra khỏi ngăn xếp.

Biến bản ghi "mark" : với các thành phần x và y tương ứng với tọa độ x và y của con trỏ, vì vậy biến này được sử dụng trong quá trình vẽ khi gặp lệnh 'f', có thể đoán được rằng sau khi vẽ sẽ là công đoạn cập nhật tọa độ mới cho con trỏ. Thành phần "dir" chỉ hướng di chuyển hiện thời của con trỏ, được sử dụng trong quá trình vẽ khi gặp lệnh 'f', và được cập nhật khi gặp lệnh 't'.

Biến "dx" and "dy" : vai trò của các biến này chưa được rõ ràng với những thông tin đã có. Qua tên biến, có thể dự đoán chúng được sử dụng trong quá trình vẽ khi gặp lệnh 'f', và vai trò của các biến này sẽ được làm rõ trong phần sau.

Hình 1 tóm tắt lại các giải thích trên. Để dễ dàng hình dung hoạt động của biến "opno", mảng "insts" được biểu diễn thẳng đứng như trong hình.



Hình 1: Vai trò và nội dung của các mảng và biến được sử dụng

(Tóm tắt cấu trúc chương trình)

Đoạn quan trọng nhất của hàm "execute" được thực hiện bên trong một vòng lặp "while-do" với điều kiện kết thúc là khi giá trị của insts["opno].code khác '\0'. Bên trong vòng lặp "while-do" là một câu lệnh rẽ nhánh "switch-case" với 4 nhánh tương ứng với giá trị của insts["opno].code và khoảng trắng G. Trong câu lệnh switch-case, có bốn đoạn xử lý [1][2][3][4], mỗi đoạn tương ứng với một quá trình được chỉ ra trong mã lệnh. Khoảng trắng G nằm bên ngoài câu lệnh switch-case, do đó nó sẽ được thực hiện sau khi tất cả các lệnh được lưu trong mảng đã hoàn thành. Dựa vào điều này, có thể đoán rằng khoảng trắng G chính là quá trình tăng giá trị của biến opno lên 1 đối với mảng "insts", điều này sẽ được khẳng định sau khi hiểu rõ nội dung cấu trúc của cả chương trình.

Cấu trúc của hàm "execute" rất đơn giản. Mặc dù vậy, để tránh lỗi và nhầm lẫn, có một lời khuyên là nên tự mình thêm vào các dòng phân chia chú thích như trên khi làm bài. Ví dụ, khoảng trắng G ngay lập tức được thấy bên ngoài câu lệnh switch-case. Từ đó, đáp án đúng cho G cũng dễ dàng được suy ra.

(Nội dung chương trình)

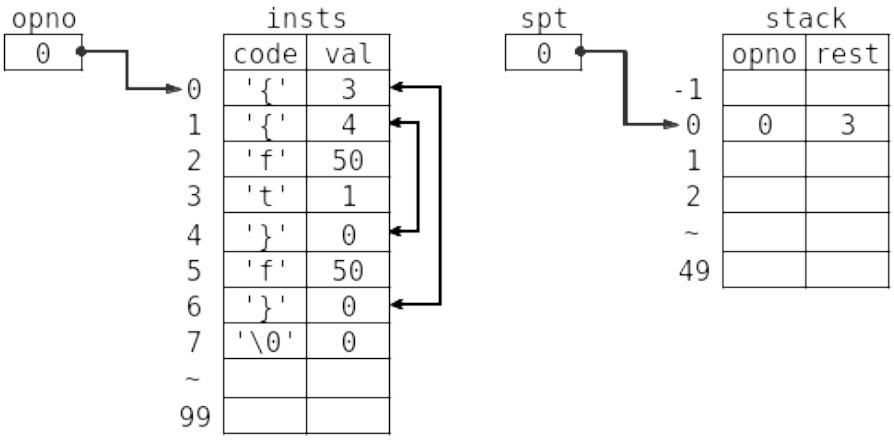
Sau khi hiểu rõ các mảng và biến được sử dụng cũng như cấu trúc chương trình, nhiệm vụ tiếp theo là điền vào các chỗ trống trong chương trình dựa trên các dữ liệu mẫu được cho trong câu hỏi. Các ô trống được xử lý theo thứ tự thực hiện các lệnh lưu trong mảng trong dữ liệu mẫu. Phần tử đầu tiên của mảng là lệnh '{', vì thế chúng ta bắt đầu từ lệnh trong phần 1, tương ứng với ô trống A.

Ô trống A:

Ô trống A chứa chỉ số của mảng "stack". Như được ghi chú tại dòng thứ 5 dành cho biến "spt" và từ các dòng sau ô trống, chỉ số của mảng "stack" phải là "spt". Nhưng vì giá trị khởi tạo của biến "spt" là -1, nên ta chưa thể sử dụng ngay biến này. Việc tăng chỉ số này lên là cần thiết trước khi kiểm tra giá trị của biến "opno", một thành phần của "stack". Do đó, ô trống A cần được điền (a) ++spt.

Hình 2 dưới đây mô tả rõ hơn giá trị mã lệnh của phần tử đầu tiên

trong dữ liệu mẫu, được xử lý qua dòng 14.



Hình 2: Trạng thái khi '{' trong phần tử đầu tiên được xử lý qua dòng 14

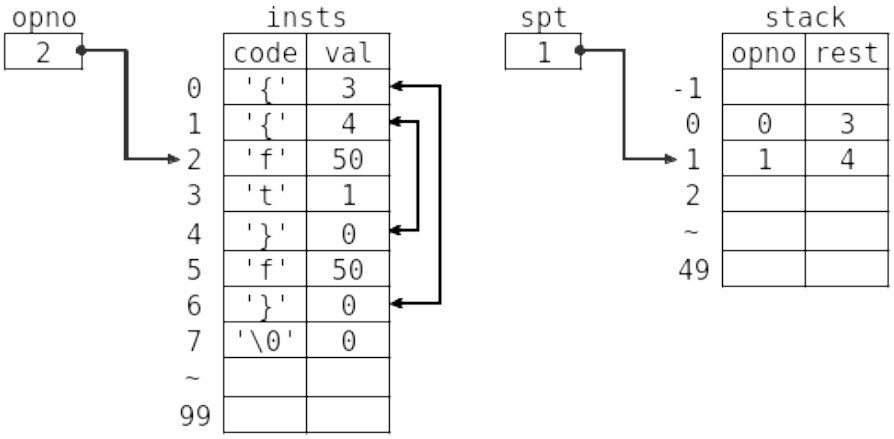
Sau Hình 2, trong dòng thứ 15, chương trình thoát khỏi câu lệnh rẽ nhánh, bỏ qua các trường hợp "`switch`" khác và chuyển tới dòng 38, chứa khoảng trắng G.

Ô trống G:

Khi chúng ta nghiên cứu cấu trúc chương trình, một điều nhận thấy rằng ô trống G luôn được thực hiện sau mỗi lần câu lệnh lưu trong mảng "insts" được xử lý. Dự đoán rằng giá trị của biến chỉ số "opno" sẽ được tăng lên tại đây. Ta sẽ khẳng định điều đó với dữ liệu mẫu.

Hình 2 là một ví dụ trạng thái ngay trước khi thực hiện lệnh trong ô trống G. Nếu không có gì được thực hiện tại đây và chương trình tiếp tục tới các dòng 10 và 11, giá trị của biến "opno" sẽ luôn là 0, do đó '{' tại câu lệnh đầu tiên, đã được xử lý, sẽ tiếp tục được xử lý. Nhưng, thực tế, '{' tại phần tử thứ 1 cần được xử lý. Nói cách khác, chỉ số "opno" cần phải được tăng lên trong ô trống G. Ta có thể khẳng định "opno++" là đáp án của ô trống G. Đáp án do đó là (c).

Hình 3 dưới đây chỉ ra trạng thái sau khi lệnh '{' tại phần tử thứ 1 được xử lý qua dòng 38 và "opno" tăng lên nhận giá trị là 2.



Hình 3: Lệnh '{' tại phần tử thứ 1 được xử lý qua dòng 38 và giá trị của "opno" được tăng lên nhận giá trị 2.

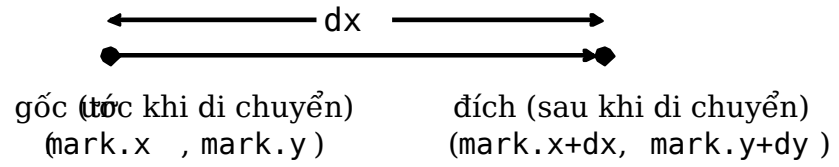
Nếu chương trình tiếp tục, nội dung lệnh tiếp theo "insts[opno].code" sẽ là 'f', chương trình sẽ thực hiện tới khối lệnh [3], tức là các ô trống C và D. Các dòng này gần như tương tự nhau. Ta sẽ điền C trước và từ đó suy ra giá trị của ô trống D.

Ô trống C:

Trong dòng 21, chứa ô trống C, giá trị của biểu thức sử dụng toán tử điều kiện "?" được gán cho biến "dx". Do do, một cách tự nhiên, ta sẽ xem xét vai

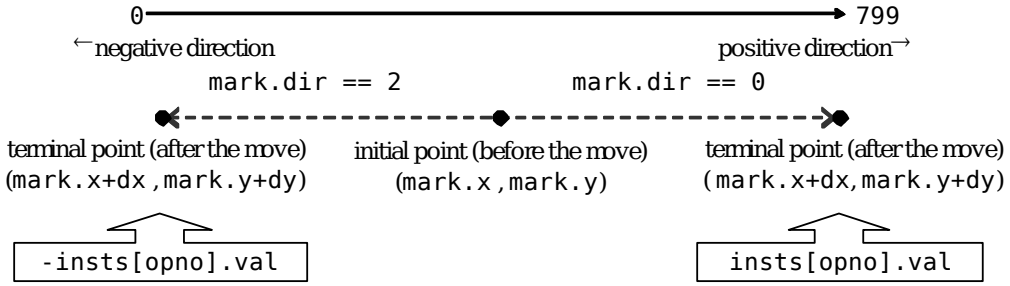
trò của biến "dx" trước.

"dx" được sử dụng trong dòng 24 và 25. Trong dòng 24, nó được sử dụng làm tham số cho hàm "drawLine", làm nhiệm vụ vẽ một đường thẳng. Trong dòng 25, giá trị này được cộng thêm vào "mark.x". Bởi vì dòng 27 sử dụng hàm "paintMarker" để vẽ con trỏ, ta hiểu rằng việc này là để cập nhật giá trị tọa độ x của con trỏ. Do đó, ta suy ra được dx chính là khoảng cách giữa tọa độ x của con trỏ trước khi di chuyển và tọa độ x của nó sau khi di chuyển. Nói cách khác, biến dx là khoảng cách trục hoành khi di chuyển, như minh họa dưới đây.



Hình 4: Vai trò của biến "dx" (Phần 1)

Mặc dù vậy, dựa vào câu hỏi, khoảng cách di chuyển là "insts[opno].val". Điều này có nghĩa là tọa độ x của điểm đích sẽ nhận giá trị "mark.x + insts[opno].val". Khi chúng ta tiếp tục xem xét, một điều nhận thấy là khi di chuyển về phía trái, ta cần "insts[opno].val" từ "mark.x". Nói cách khác giá trị của biến "dx" khi đó sẽ có trị tuyệt đối bằng "insts[opno].val" nhưng mang dấu âm. Hình vẽ sau giải thích cụ thể điều này:



Hình 5: Vai trò của biến "dx" (Phần 2)

Sauk hi xem xét tất cả các vấn đề trên, ta sẽ tìm công thức cho chúng ta giá trị của biến "dx".

Toán tử điều kiện "?" được sử dụng để tìm giá trị của biến "dx". Toán tử này được sử dụng "công thức 1 ? công thức 2 : công thức 3". Nếu công thức 1 nhận giá trị đúng, công thức 2 sẽ được thực hiện và giá trị của nó được trả lai; ngược lại nếu 1 sai, công thức được thực hiện sẽ là công thức 3. Ở đây, công thức 1 là "mark.dir %2 ==0", chỉ nhận giá trị đúng khi "mark.dir" nhận giá trị 0 (phải) hoặc 2 (trái). Do đó, công thức 2 được thực hiện và gán cho biến "dx" khi con trỏ di chuyển sang phải hoặc trái, công thức 3 được thực hiện khi con trỏ di chuyển lên hoặc xuống.

"dx" chứa khoảng cách trục hoành, vì thế khi di chuyển lên xuống, giá trị này bằng 0. Điều này thu hẹp các đáp án khả dĩ xuống, chỉ giữ lại các trường

hợp mà công thức 3 là 0: (a)-(d) và (j). Từ những kết quả còn lại này, ta sẽ lựa chọn ra đáp án có giá trị của công thức 2 là "insts[opno].val" khi di chuyển sang phải (mark.dir nhận giá trị 0) và "insts[opno].val" khi di chuyển sang trái (mark.dir nhận giá trị 2). Điều kiện này thỏa mãn với "(1 − mark.dir) * insts[opno].val" khi "mark.dir" là 0 (phải) và "1 * insts[opno].val" khi "mark.dir" là 2 (trái). Cuối cùng, (a) là kết quả của ô trống C.

Ta sẽ tổng kết lại quá trình suy luận để từ đó có thể sử dụng lại suy luận này với ô trống D.

Biến "dx" là khoảng cách trục hoành khi di chuyển, nhận giá trị dương "insts[opno].val" khi "mark.dir" là 0 (phải), và giá trị âm "insts[opno].val" khi "mark.dir" là 2 (trái), và giá trị 0 khi "mark.dir" là 1 hoặc 3 (lên hoặc xuống). Đáp án sau cho ô trống C là hợp lý.

```
dx = ( mark.dir % 2 == 0 ) ? (1 − mark.dir) * insts[opno].val : 0
```

Ô trống D:

Ta sử dụng lại các suy luận đã dùng với ô trống C.

Biến "dy" là khoảng cách trục tung khi di chuyển, mark.dir. Khi "mark.dir" là 0 (phải) hoặc 2 (trái), biến này nhận giá trị 0. Nếu "mark.dir" là 1 (lên), nó nhận giá trị âm "insts[opno].val". Khi "mark.dir" là 3 (xuống), nó nhận giá trị dương "insts[opno].val".

Do đó công thức 2 trong biểu thức điều kiện phải là 0. Đáp án đúng cho D phải là các đáp án từ (e)~(i). Công thức 3 sẽ là "1 * insts[opno].val" khi "mark.dir" là 1 (lên) và "1 * insts[opno].val" khi "mark.dir" là 3 (xuống). Cuối cùng, công thức sau là đáp án hợp lý.

```
dy = ( mark.dir % 2 == 0 ) ? 0 : (mark.dir
insts[opno].val
```

Do đó, (h) là đáp án đúng cho D.

Sau khi 'f' tại câu lệnh có chỉ số 2 được xử lý ở khối mã [3], dòng 38 tăng giá trị của biến opcode lên 1 so với giá trị 3. Vì vậy, "insts[opno].code" trở thành 't', khối mã [2] được thực hiện, chứa ô trống B.

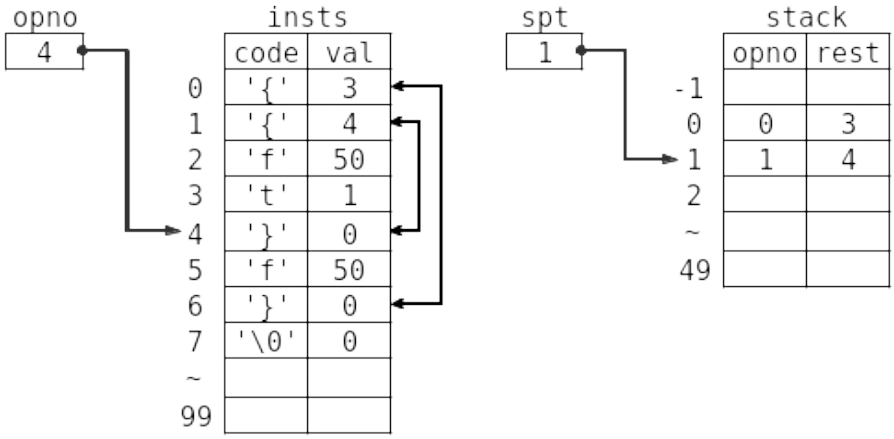
Ô trống B:

Theo câu hỏi, nếu mã lệnh "insts[opno].code" là 't', hướng di chuyển của con trỏ sẽ lệch đi 90 độ × val ngược chiều kim đồng hồ. Nói cách khác, ô trống B tính toán hướng di chuyển mới của con trỏ sau khi gặp lệnh chuyển

hướng. Giá trị chỉ hướng là các số nguyên từ 0~3, do đó ta cần chọn một công thức chỉ cho kết quả từ 0~3. Trong số các đáp án, công thức duy nhất thỏa mãn điều kiện này là `( mark.dir + insts[opno].val ) % 4`. Cuối cùng, (c) là đáp án đúng của B.

Ví dụ, ta thấy (g) không phải là đáp án đúng. Nếu `mark.dir` là 3 và `"insts[opno].val"` là 5, kết quả sẽ là 3 + 5 % 4 = 3 + 1 = 4, không phải là hướng có thể.

Trong hình 6, giải thích khi lệnh `t'` của phần tử với chỉ số 3 được xử lý trong khối mã [2] và chỉ số `"opno"` được tăng lên, nhận giá trị 4 trong dòng 38.

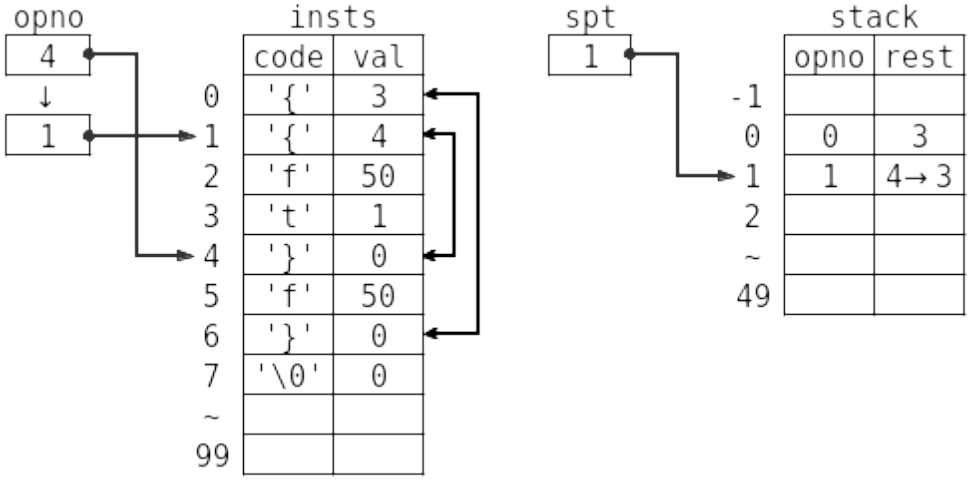


Hình 6: Chỉ số "opno" được tăng lên trong dòng 38 và nhận giá trị 4

Tới thời điểm này, mã lệnh "`insts[opno].code`" là '`}`', vì vậy chương trình sẽ thực hiện khối mã [4], chứa các ô trống E và F.

Ô trống E:

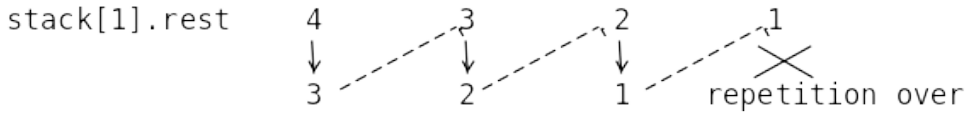
Khối mã [4], chứa ô trống E, được xử lý khi giá trị của "`insts[opno].code`" là '`}`', vì thế có thể dự đoán đây sẽ là quá trình xử lý việc lặp đoạn mã giới hạn bởi '`{`' và '`}`'. Điều này có thể khẳng định qua dòng 31 và 32. Ví dụ, trạng thái trong hình 6, nếu dòng 31 được thực hiện, giá trị của biến chỉ số "`opno`" thay đổi từ 4 thành 1, chỉ số của lệnh '`{`' tương ứng cặp với lệnh '`}`' hiện tại. Chỉ số "`opno`" được sử dụng để truy nhập mảng "`insts`"; nó luôn được tăng thêm 1 tại dòng 38 mỗi khi có một câu lệnh được thực hiện. Do đó, gán giá trị của biến "`opno`" một giá trị cụ thể sẽ chuyển việc thực hiện tiếp theo tới lệnh mang giá trị đó. Nói cách khác, các lệnh nằm trong cặp '`{`' và '`}`' sẽ được thực hiện lại (lặp). Khi dòng 32 được xử lý, giá trị của thành phần "`rest`" (số lần lặp) trong "`stack`" sẽ được giảm xuống.



Hình 7: Trạng thái sau khi dòng 31 được thực hiện

Như thấy ở đây, dòng 31 và 32 được dành cho việc xử lý lặp, do đó câu lệnh "`if`" trong dòng 30, chứa ô trống E, phải là điều kiện cho việc lặp. Ta sẽ phải kiểm tra giá trị của "`tack[stp].rest`" để tìm ra điều kiện khi nào việc lặp được tiếp tục.

Giá trị ban đầu của "`stack[stp].rest`" là 4, và dòng 32 giảm giá trị này đi 1 mỗi lần được thực hiện. Như đã trình bày ở trên, sau khi dòng này được thực hiện, chương trình sẽ thực hiện việc lặp. Cụ thể hơn nếu có 4 lần lặp, giá trị này sẽ giảm từ 4 xuống 3 sau khi thực hiện xong chu trình đầu tiên và chu trình thứ 2 bắt đầu, giá trị này giảm xuống 2 khi chu trình thứ 3 bắt đầu và 1 khi chu trình thứ 4 bắt đầu. Do đó, ngay trước khi thực hiện lần lặp thứ 4, giá trị này là 1. Sau khi thực hiện xong chu trình này, việc lặp đã kết thúc; dòng thứ 34 sẽ được thực hiện.

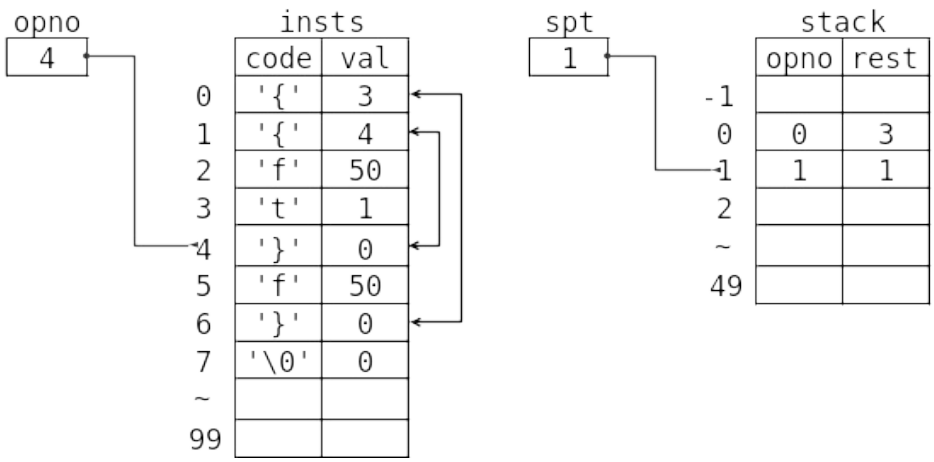


Hình 8

Tổng kết lại. Ta cần điền vào ô trống G một biểu thức điều kiện sao cho khi giá trị của "`stack[stp].rest`" lớn hơn 1, biểu thức này nhận giá trị đúng và dòng 31, 32 được thực hiện. Cuối cùng, đáp án (e)>1" là đáp án đúng của ô trống E.

Ô trống F:

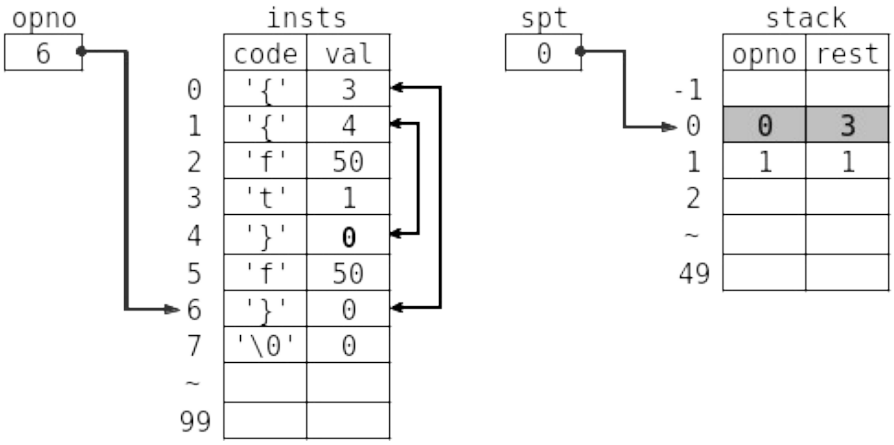
Được thực hiện khi mã lệnh "`insts[opno].code`" là '}' và "`stack[spt].rest`" nhận giá trị 1, khi vòng lặp kết thúc. Ví dụ, các lệnh nằm trong '{' tại phần tử thứ 1 và '}' tại phần tử thứ 4 được lặp 4 lần, khi ô trống F trong dòng 34 được thực hiện, mảng "`insts`" và "`stack`", cùng với các chỉ số của nó như sau.



Hình 9: Mảng "insts" và "stack" cùng với chỉ số tương ứng

Để tìm ra đáp án cho F, thử đặt ra vấn khi ta tiếp tục chương trình mà không thực hiện gì trong ô trống F

Giả sử không có gì được thực hiện từ trạng thái ở hình 9 tới quá trình tiếp theo. Đầu tiên, biến "opno" có chỉ số tăng từ 4 lên 5, mã "insts[opno].code" sẽ là 'f'. Khối mã [3] sẽ được thực hiện. Không có gì đặc biệt xảy ra tại khối mã này. Sau khi hoàn thành khối mã [3], chỉ số "opno" tăng từ 5 lên 6 tại dòng 38. Lúc này, "insts[opno].code" là '}', chương trình vì thế sẽ chuyển tới thực hiện khối mã [4]. Lúc này, nếu giá trị của biến chỉ số "spt" của mảng "stack" vẫn là 1 thì vấn đề sẽ xảy ra. Nguyên nhân bởi vì vòng lặp giới hạn bởi phần từ thứ 1 và phần tử thứ 4 đã được thực hiện xong, do đó nó cần phải được loại bỏ khỏi stack. Nói cách khác, chỉ số "spt" cần phải có giá trị 0. Điều này được giải thích cụ thể như sau.



Hình 10: Chỉ số "spt"

Để tạo ra trạng thái như trong hình 10 khi chỉ số "opno" là 6, ta cần giảm giá trị của biến chỉ số "spt" sau khi những lệnh giới hạn bởi "{" tại lệnh thứ 1 và '}' tại lệnh số 4 đã được lặp đủ 4 lần (Hình 9). Cuối cùng, (f) "spt--" cần phải là kết quả của ô trống F. Nó không thực sự loại bỏ các giá trị ra khỏi stack mà chỉ tương đương với việc thay đổi chỉ số chỉ vị trí của đỉnh stack và khi một lệnh '{' tiếp theo được xử lý, "++spt" sẽ trỏ vào đỉnh mới của stack.

Một phương pháp để điền vào F là tìm kiếm một vị trí thích hợp để giảm giá trị của chỉ số stack "spt", bởi vì chưa có câu lệnh nào giảm giá trị của nó. Thời điểm cần giảm chỉ số này là ngay khi "stack[spt].rest" hoàn thành vai trò của mình hay khi vòng lặp kết thúc. Do đó F là vị trí thích hợp để thực hiện điều này.

Câu hỏi 4 — Lập trình Java

Q4. Đọc mô tả chương trình và chương trình Java dưới đây rồi trả lời Câu hỏi con.

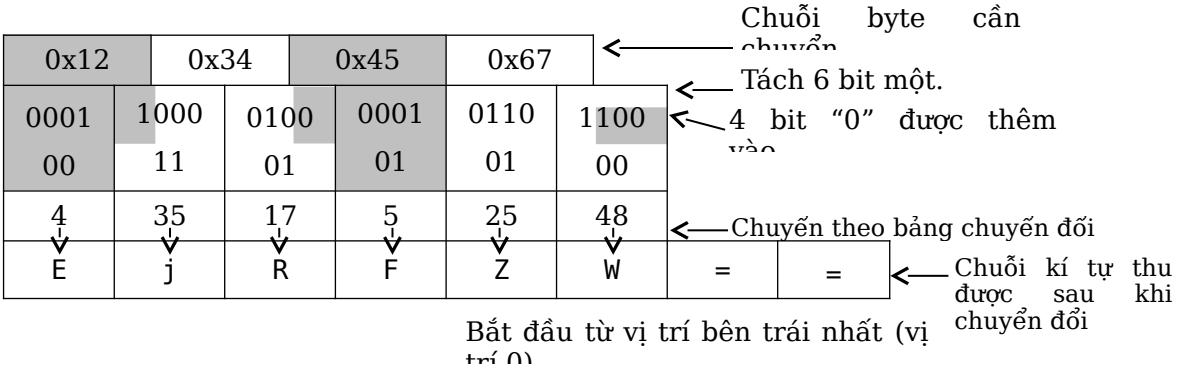
[Mô tả chương trình]

Chương trình bao gồm: a) lớp `Encoder`, có nhiệm vụ chuyển đổi chuỗi byte dữ liệu nhị phân theo một thuật toán nào đó và trả về kết quả là một chuỗi kí tự; b) một lớp chạy thử cho lớp `Encoder`. Thuật toán chuyển đổi như sau:

(1) Mỗi lần trích ra 6 bit từ đầu chuỗi byte và chuyển nó thành kí tự tương ứng theo bảng chuyển đổi. Nếu số lượng byte không phải là bội của 3 thì thêm 4 bit hoặc 2 bit "0" vào cuối chuỗi sao cho tổng số bit vẫn là bội của 6.

(2) Một số các kí tự "=" được thêm vào cuối chuỗi kí tự đã chuyển đổi sao cho tổng số kí tự là bội nhỏ nhất của 4 và lớn hơn hoặc bằng 4/3 lần số byte đã cho.

(3) Một ví dụ về quá trình chuyển đổi được thể hiện ở Hình 1 dưới đây



Hình 1. Ví dụ về quá trình chuyển đổi

(4) Lớp `Encoder` có 2 phương thức: `next` và `hasNext`. Phương thức `next` trả về lần lượt từng kí tự đã chuyển đổi theo thuật toán chuyển đổi. Nếu phương thức này được gọi khi không có kí tự nào trả về thì phát sinh ra ngoại lệ `java.util.NoSuchElementException`.

Phương thức `hasNext` là `true` nếu trong lần gọi kế tiếp, phương thức `next` trả về một kí tự chứ không phát sinh ngoại lệ.

(5) Biến `CHARS` là một mảng 64 phần tử, đóng vai trò một bảng chuyển đổi số 6 bit sang một kí tự đơn lẻ. Biến `n` lưu giữ thông tin chỉ ra vị trí tiếp theo của kí tự được trả về trong lần gọi kế tiếp của phương thức `next` (vị trí của kí tự phía bên trái nhất là 0).

Biến `bin` lưu giữ chuỗi dữ liệu nhị phân cần biến đổi.

(6) Kết quả thực thi của lớp chạy thử (lớp `EncoderTest`) đối với chuỗi byte trong Hình 1 được chỉ ra trong Hình 2.

```
EjRFZw==
```

Hình 2. Kết quả chạy

[Chương trình]

```
public class Encoder {
    static final char[] CHARS = ("ABCDEFGHIJKLMNOPQRSTUVWXYZ" +
    "abcdefghijklmnopqrstuvwxyz0123456789+/").toCharArray();
    private int n = 0;
    private byte[] bin;
    public Encoder(byte[] bin) {
        if (bin == null) this.bin = new byte[0];
        else this.bin = bin;
    }
    public boolean hasNext() {
        return n < [ A ];
    }
    public char next() {
        char letter;
        int pos = (int)(n * 0.75);
        // Lấy được 2 byte dữ liệu gồm 6 bit dành cho việc chuyển đổi
        if (pos < bin.length) {
            int cell = bin[pos++] << 8;
            if (pos < bin.length) cell += bin[pos] & 255;
            // Lấy 6 bit cần chuyển đổi và chuyển đổi chúng thành ký tự tương
ứng
            letter = CHARS[(cell >> (n + 3) % 4 * 2 + 4) & 63];
        } else {
            if ([ B ])
                throw new java.util.NoSuchElementException();
            else letter = [ C ];
        }
        n++;
        return letter;
```

```
    }
}

class EncoderTest {
    public static void main(String[] args) {
        byte[] bin = {0x12, 0x34, 0x45, 0x67};
        Encoder encoder = new Encoder(bin);
        while (encoder.hasNext()) {
            System.out.print(encoder.next());
        }
        System.out.println();
    }
}
```

Câu hỏi con

Từ các nhóm câu trả lời sau, chọn câu trả lời đúng để chèn vào các ô trống ☐ trong chương trình trên.

Nhóm câu trả lời cho A:

a) `(bin.length + 2) * 4 / 3`
b) `(bin.length + 2) / 3 * 4`
c) `bin.length * 4 / 3`
d) `bin.length / 3 * 4`

Nhóm câu trả lời cho B:

a) `!hasNext( )`
b) `hasNext( )`
c) `n < bin.length`
d) `n >= bin.length`

Nhóm câu trả lời cho C:

a) `‘ ’`
b) `‘=’`
c) `CHARS[n]`
d) `next( )`

Đáp án câu 4

Chuyển đổi dữ liệu nhị phân

Đáp án

A – b, B – a, C – b

Giải thích

Đây là một chương trình đọc chuỗi byte dữ liệu nhị phân 6 bit một lần và chuyển đổi thành chuỗi kí tự tương ứng. Vì đây là biểu diễn dữ liệu 6 bit với 64 kiểu kí tự nên kiểu chuyển đổi này được gọi là “mã hóa BASE64”. Vì dữ liệu nhị phân có thể được biểu diễn bởi các kí tự của bảng mã ASCII nên chúng được sử dụng để truyền và nhận dữ liệu nhị phân chủ yếu trong thư điện tử. Chương trình gồm một lớp `Encoder` không có bất kì chỗ nào khó hiểu về mặt cú pháp, nhưng có một vài khái niệm phức tạp, như là tính toán số lượng kí tự hay đọc chuỗi byte 6 bit một lần. Hơn nữa, chỉ có 3 ô trống nên mỗi ô trống trong câu hỏi có điểm nhiều hơn; vì thế phải rất cẩn thận để không mắc những lỗi do sơ suất.

[Mô tả chương trình]

Hằng `"CHARS"` được định nghĩa trong lớp `Encoder` là một bảng dùng cho chuyển đổi chuỗi 6 bit lấy từ dữ liệu nhị phân thành các kí tự. Một chuỗi kí tự trong Java là một tham chiếu tới một đối tượng `String`, nên có thể sử dụng một số phương thức như `"toCharArray"`, như đã thấy trong chương trình.

Trong lớp `Encoder`, chương trình nhận chuỗi byte dữ liệu nhị phân với một phương thức khởi tạo (constructor) và trả về mỗi lần 1 kí tự bởi phương thức `"next"`. Phương thức `"hasNext"` trả về giá trị kiểu `boolean`, thể hiện có kí tự tiếp theo hay không.

Đầu tiên, ta xem xét quá trình chuyển đổi dữ liệu nhị phân thành các kí tự, và được thực hiện bởi phương thức `"next"`, mặc dù để trả lời được câu hỏi, có thể không nhất thiết phải hiểu nội dung được diễn giải ở đây.

Như trong đầu bài, biến `"n"` chứa vị trí kí tự được trả về tiếp theo (“0” cho kí tự đầu). Mảng dữ liệu nhị phân `"bin"` chứa 8 bit cho mỗi phần tử, chia thành các phần 6 bit, và chuyển đổi thành các kí tự đầu ra, nên số kí tự đầu ra sẽ không hoàn toàn tương ứng với số phần tử (chỉ số) của mảng `"bin"`. Do đó, nhân `"n"` với 0.75 (= 6 / 8) sẽ thu được chỉ số phần tử trong mảng `"bin"` tương

ứng với "n", và được lưu giữ trong biến "pos". Ở đây, khi gọi (int)là ta đã cắt bỏ phần thập phân của kết quả tính toán.

Vấn đề ở đây là trong dữ liệu 8 bit tương ứng với chỉ số thu được từ biến "pos", không phải lúc nào cũng bao gồm dữ liệu 6 bit cần thiết. Ví dụ, 6 bit tương ứng với kí tự đầu tiên rõ ràng là 6 bit đầu của phần tử có chỉ số bằng 0. Vậy dữ liệu tương ứng với kí tự tiếp theo (n = 1) thì sao? Đương nhiên, đó là 6 bit dữ liệu tiếp theo, nhưng trong số 8 bit của phần tử có chỉ số 0, chỉ còn thừa lại 2 bit. Do đó ta cần phải lấy tiếp 4 bit đầu của phần tử có chỉ số 1. Nói cách khác, khi n = 1, giá trị của biến "pos" là phần nguyên (cắt bỏ phần thập phân) của 1 × 0.75 = 0.75, là 0. Do đó, giá trị của "pos" thu được từ biến "n" thực ra là chỉ số của phần tử chứa bit đầu của chuỗi 6 bit đang xét, và 6 bit đang xét đó có thể không hoàn toàn nằm trong dữ liệu biểu diễn bởi chỉ số. Bởi vậy, nếu giá trị của "pos" nhỏ hơn "bin.length", đoạn xử lí sau được sử dụng để lấy 16 bit dữ liệu cần thiết vào biến “cell” từ mảng "bin".

```
int cell = bin[pos++] << 8;
if (pos < bin.length)  cell += bin[pos] & 255;
```

Ở đây, chú ý đến phần xử lí "bin[pos++] << 8", bởi "bin" là biến mảng kiểu byte (8-bit) và bình thường nếu dịch trái 8 bit đối với dữ liệu 8 bit sẽ cho kết quả = 0. Tuy nhiên, với toán tử dịch bit (shift) trong Java, dù biến thuộc kiểu byte hay kiểu short (16-bit), mọi biến trước hết được chuyển đổi sang kiểu int (32-bit) và sau đó phép xử lí mới được thực hiện (nếu kết quả được lưu vào không gian nhỏ hơn 32 bit thì các bit cao bị cắt bỏ). Nếu tiếp sau vẫn còn dữ liệu (pos < bin.length), thì 8 bit của phần dữ liệu tiếp được thêm vào (cell += bin[pos] & 255) sao cho dữ liệu 16 bit có thể được lưu trong biến"cell". Phép xử lí bit "AND" với 255 (1111 1111 trong hệ nhị phân) được thực hiện trước phép cộng đơn giản chỉ để cộng với 8 bit thấp nhất của "bin [pos]" (trong trường hợp này, "bin[pos]" chỉ có 8 bit nên điều này không cần thiết, nhưng thông thường kiểu phép toán bit này hay được sử dụng). Sau khi đã có dữ liệu 16-bit tương ứng với vị trí kí tự được chỉ ra bởi biến "n" được lưu trong biến "cell", ta chuyển sang phần tiếp theo:

```
letter = CHARS[(cell >> (n + 3) % 4 * 2 + 4) & 63];
```

Chú ý phần "(cell >> (n + 3) % 4 * 2 + 4) & 63" ở dòng trên. Bằng sự tính toán này ta thu được 6 bit sẽ được chuyển đổi, và trở thành chỉ số của mảng "CHARS" đồng thời tìm ra được kí tự tương ứng.

Xem xét công thức "(cell >> (n + 3) % 4 * 2 + 4) & 63" một lần nữa. Thông thường, phép toán được chia làm 4 trường hợp tùy thuộc vào giá trị của n, nhưng ở đây 4 trường hợp được kết hợp trở thành một. Ví dụ, xem xét dữ liệu nhị phân dưới dây. Chú ý rằng phần "& 63" là để lấy về 6 bit thấp nhất bằng cách lấy "&" với $63_{10}=111111_2$.

	[0]	[1]	[2]	[3]	
Dữ liệu của mảng "bin"	0x12	0x34	0x45	0x67	.....
(số nhị phân)	00010010	00110100	01000101	01100111	.....

Ba khối dữ liệu 8 bit cho ta chính xác 4 khối dữ liệu 6 bit, nên 3 khối dữ liệu 8 bit đầu tiên trong bin[0], bin[1], và bin[2]

pos =	0	1	2
	00010010	00110100	01000101

cho 4 khối dữ liệu 6 bit (kí tự):

n =	0	1	2	3
	000100	100011	010001	000101

[1] Khi n=0

"cell" chứa chuỗi bit 000100 10 00110100 là kết quả ghép nối của bin[0] và bin[1]. Để thu được 6 bit trong hộp, ta dịch phải chuỗi trên đi 10 bit và lấy 6 bit thấp nhất.

(0 + 3) % 4 * 2 + 4 = 3 * 2 + 4 = 10

[2] Khi n=1

"cell" chứa chuỗi bit 000100 10 0011 0100 là kết quả ghép nối của bin[0] và bin[1]. Để thu được 6 bit trong hộp, ta dịch phải chuỗi trên đi 4 bit và lấy 6 bit thấp nhất.

(1 + 3) % 4 * 2 + 4 = 0 * 2 + 4 = 4

[3] Khi n=2

Lúc này `pos = 1` nên "`cell`" chứa chuỗi bit 0011 0100 01 000101 là kết quả ghép nối của `bin[1]` và `bin[2]`. Để thu được 6 bit, ta dịch phải chuỗi trên đi 6 bit và lấy 6 bit thấp nhất

(2 + 3) % 4 * 2 + 4 = 1 * 2 + 4 = 6

[4] Khi n=3

Vì `pos = 2` nên "`cell`" chứa chuỗi bit 01 000101 01100111 là kết quả ghép nối của `bin[2]` và `bin[3]`. Để thu được 6 bit, ta dịch phải chuỗi trên đi 8 bit và lấy 6 bit thấp nhất.

(3 + 3) % 4 * 2 + 4 = 2 * 2 + 4 = 8

Với `n>=4`, 4 trường hợp xử lí trên (`n=0-3([1]~[4])`) được lặp lại, vì thế điều kiện `n=0~3` có thể thay thế bởi `n % 4 = 0~3`. Như vậy, có thể tóm tắt về số bit biến "`cell`" cần dịch phải như sau:

`n % 4`	Số bit biến "`cell`" dịch phải
0	10
1	4
2	6
3	8

Biến đổi sang dạng (+4) giống trong công thức, ta có:

`n % 4`	Số bit biến "`cell`" dịch phải
0	6 + 4
1	0 + 4
2	2 + 4
3	4 + 4

Phân tích thành thừa số 2 cho phần đầu trong biểu thức, ta có:

`n % 4`	Số bit biến "`cell`" dịch phải
0	3 * 2 + 4
1	0 * 2 + 4
2	1 * 2 + 4
3	2 * 2 + 4

Do đó, tổng số bit biến "`cell`" cần dịch có thể tính được nhờ công thức "`(n + 3) % 4 * 2 + 4`". Khi biến "`cell`" được dịch phải một số bit thích hợp, ta thực hiện phép toán bit AND giữa kết quả thu được với 63. Bằng cách đó, ta có thể thu được 6 bit cần thiết.

[Câu hỏi con]

Ô trống A

Xem xét điều kiện để "hasNext" trả về true. Câu hỏi đưa ra mô tả sau: "Một số các kí tự “=” được thêm vào cuối chuỗi kí tự đã chuyển đổi sao cho tổng số kí tự là bội nhỏ nhất của 4 và lớn hơn hoặc bằng 4/3 lần số byte đã cho". Do đó, trong khi điều này còn đúng, các kí tự "=" tiếp tục được đưa ra thậm chí cả khi không còn dữ liệu nhị phân nào để chuyển đổi thành kí tự. Sau đó, lớp "EncodeTest" xuất ra các kí tự trong khi "hasNext" là true, vì thế ngay khi số kí tự thu được bởi phương thức "next" nhỏ hơn "bội nhỏ nhất của 4 và lớn hơn hoặc bằng 4/3 lần số byte đã cho”, "hasNext" phải trả về true. Nói cách khác, biến "n" xuất hiện trước ô trống A biểu diễn số kí tự đã thu được bởi phương thức "next". Do đó, ta cần một công thức tính "bội nhỏ nhất của 4 và lớn hơn hoặc bằng 4/3 lần số byte đã cho" cho ô trống A. Bạn phải cẩn thận, trong Java, phép chia một số nguyên bởi một số nguyên, phần thập phân bị cắt bỏ, chỉ đưa ra kết quả cuối cùng là số nguyên. Từ đó có thể tự tìm câu trả lời. Câu trả lời đúng là (b) "(bin.length + 2) /3 * 4".

Lí do thêm 2 vào "bin.length" trước khi chia cho 3 là vì thương chính xác bằng bin.length / 3 nếu "bin.length" là bội của 3 và thương bằng "bin.length / 3 + 1" trong các trường hợp khác. Nhân thương này với 4 cho ta chính xác " bội nhỏ nhất của 4 và lớn hơn hoặc bằng 4/3 lần số byte đã cho".

Ở các lựa chọn khác, (a) và (c) tiến hành nhân 4 trước khi chia cho 3, nên kết quả không thể là bội của 4. (d) sai vì kết quả nhỏ hơn 4/3 lần số byte đã cho.

Ô trống B

Ở đây ta cần tìm điều kiện để khi gọi phương thức "next", ngoại lệ java.util.NoSuchElementException được ném ra. Liên quan tới ngoại lệ là mục (4) của [Mô tả chương trình] trong phần câu hỏi. Ngoại lệ tương ứng được phát sinh "nếu phương thức này được gọi mà không có kí tự nào trả về...". Khi dữ liệu nhị phân được chuyển đổi (pos < bin.length), phương thức "next" chuyển đổi dữ liệu thành kí tự và trả về kí tự thu được. Ngay cả khi không thể chuyển đổi dữ liệu, nó phải tiếp tục xuất ra kí tự "=" cho đến khi đạt được số kí tự yêu cầu vì trong phần câu hỏi có viết: " Một số các kí tự “=” được thêm vào cuối chuỗi kí tự đã chuyển đổi sao cho tổng số kí tự là bội nhỏ nhất của 4 và lớn hơn hoặc bằng 4/3 lần số byte đã cho". Xem xét trong nhóm câu trả lời, không có câu nào thực hiện tính toán như thế. Tuy nhiên,

với phần chú giải cho ô trống A ở trên, "`hasNext`" trả về một giá trị kiểu `boolean` cho biết đã đạt được tổng số kí tự yêu cầu hay chưa bằng cách tính toán tổng số kí tự. Ta sẽ sử dụng điều này. Nói cách khác, ta sử dụng thực tế là có các kí tự được trả về khi mà `hasNext` còn đúng. Ô trống B là điều kiện để ném ra ngoại lệ, do đó ta lấy phủ định của "`hasNext`"; câu trả lời đúng là (a) "`!hasNext()`".

Ô trống C

Như đã giải thích cho phần ô trống A và B, phương thức "`next`" tiếp tục xuất ra các kí tự "=" cho đến khi đạt được số kí tự yêu cầu, dù sau khi dữ liệu byte đã được chuyển đổi đã hết. Ô trống C chứa kí tự đầu ra cho mục đích đó. Chính vì thế, câu trả lời đúng là (b) '='.

Câu hỏi 5 Chương trình Java

Q4. Đọc mô tả chương trình và chương trình Java dưới đây rồi trả lời Câu hỏi con.

[Mô tả chương trình]

Giao diện `CharIterator` định nghĩa một thao tác trích rút lần lượt các kí tự (`char`) từ thể hiện của nó, độc lập với cấu trúc dữ liệu. Trong `CharIterator`, các phương thức sau được định nghĩa:

`public char next()`

Nếu kí tự tiếp theo tồn tại thì trả về ký tự đó. Nếu không tồn tại, ngoại lệ `java.util.NoSuchElementException` sẽ được ném ra. Ví dụ như nếu `CharIterator` có *n* kí tự, thì kí tự đầu tiên được trả về cho lần gọi đầu tiên, và kí tự thứ 2 được trả về cho lần gọi thứ 2. Bằng cách này, các kí tự lần lượt được trả về cho đến khi thực hiện lần gọi thứ n. Với lần gọi thứ *n*+1 hoặc sau đó, ngoại lệ `NoSuchElementException` sẽ được ném ra. Chú ý rằng, `NoSuchElementException` là lớp con của `java.lang.RuntimeException`.

`public boolean hasNext()`

Nếu kí tự tiếp theo tồn tại thì trả về `true`. Ngược lại, trả về `false`.

Lớp `CharIteratorFactory` định nghĩa một phương thức trả về `CharIterator` phù hợp với kiểu dữ liệu của đối số.
Trong `CharIteratorFactory`, các phương thức sau được định nghĩa:

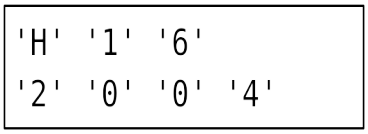
`public static CharIterator getCharIterator(String data)`

Trả về `CharIterator`, trích rút lần lượt các kí tự từ đối số kiểu `String`. Nếu đối số truyền vào là `null` thì ngoại lệ `NullPointerException` được ném ra.

`public static CharIterator getCharIterator(char[][] data)`

Trả về `CharIterator`, trích rút lần lượt các kí tự từ đối số là một mảng của mảng (mảng kí tự 2 chiều) chứa các phần tử kiểu `char`. Nếu đối số truyền vào là `null` thì ngoại lệ `NullPointerException` được ném ra. Các kí tự được trích rút theo thứ tự tăng dần của giá trị chỉ số của mảng kí tự và mảng của nó. Ví dụ, với biến m, có kiểu `char[][]`, các thành phần `m[i][j]` của nó được trích rút theo thứ tự tăng dần của `i,` và với `i` như nhau, thì theo thứ tự tăng dần của `j`.

Lớp `CharIteratorTest` là một chương trình chạy thử các phương thức được định nghĩa trong `CharIteratorFactory`. Kết quả thực thi của phương thức `main` được thể hiện ở hình dưới:



Kết quả thực thi của CharIteratorTest.main

[Chương trình 1]

```
public interface CharIterator {
    public boolean hasNext();
    public char next();
}
```

[Chương trình 2]

```
import java.util.NoSuchElementException;

public class CharIteratorFactory {
    public static CharIterator getCharIterator(String data) {
        if (data == null)
            throw new NullPointerException();
        // Tạo và trả về một thể hiện của CharIterator
        // trả về lần lượt các kí tự từ dữ liệu String.
        return A;
    }

    public static CharIterator getCharIterator(char[][] data) {
        if (data == null)
            throw new NullPointerException();
        // Tạo và trả về một thể hiện của CharIterator
        // trả về lần lượt các kí tự từ một mảng kí tự 2 chiều.
        return B;
    }
}
```

```
class StringCharIterator implements CharIterator {
    private String data;
    private int index = 0;

    StringCharIterator(String data) {
        this.data = data;
    }
    // Kiểm tra kí tự tiếp theo có tồn tại trong data hay không.
    public boolean hasNext() {
        return [          C          ];
    }

    public char next() {
        // Nếu kí tự tiếp theo không tồn tại, ném ra ngoại lệ NoSuchElementException.
        if (index >= data.length())
            throw new NoSuchElementException();
        // Trả về kí tự tiếp theo cho data và cập nhật giá trị chỉ số.
        return [          D          ];
    }
}


class Char2DArrayCharIterator implements CharIterator {
    private char[][] data;
    private int index1 = 0, index2 = 0;

    Char2DArrayCharIterator(char[][] data) {
        this.data = data;
    }
    public boolean hasNext() {
        // Nếu phần tử của data[index1][index2] tồn tại, trả về true.
        // Nếu không, tìm phần tử tiếp theo.
        // Nếu phần tử tiếp theo không tồn tại, trả về false.
        for (; index1 < data.length; index1++) {
            if (data[index1] != null
                    && index2 < data[index1].length) {
                return true;
```

```
            }
            E;
        }
        return false;
    }
    public char next() {
        // Gọi phương thức hasNext để kiểm tra phần tử tiếp theo. Nếu tồn tại,
        // phần tử đó được trả về, và cập nhật giá trị chỉ số.
        // Nếu không có phần tử nào tồn tại, ngoại lệ NoSuchElementException
được ném ra.
        if (hasNext()) {
            return F;
        }
        throw new NoSuchElementException();
    }
}
```

[Chương trình 3]

```
public class CharIteratorTest {
    public static void main(String[] args) {
        CharIterator itr =
            CharIteratorFactory.getCharIterator("H16");
        printIterator(itr);

        itr = CharIteratorFactory.getCharIterator(
                new char[][] {   { '2' },
                                 { '0' },
                                 null,
                                 { '0', '4' }});
        printIterator(itr);
    }
    private static void printIterator(CharIterator itr) {
        while (itr.hasNext()) {
            System.out.print("'" + itr.next() + "' ");
        }
        System.out.println();
    }
}
```

Câu hỏi con

Từ các nhóm câu trả lời sau, chọn câu trả lời đúng để chèn vào các ô trống ☐ trong chương trình trên.

Nhóm câu trả lời cho A và B:

a) `getCharIterator((char[][]) data)`
b) `getCharIterator((String) data)`
c) `new Char2DArrayCharIterator()`
d) `new Char2DArrayCharIterator(data)`
e) `new StringCharIterator()`
f) `new StringCharIterator(data)`

Nhóm câu trả lời cho C:

a) `index < data.length()`
b) `index <= data.length()`
c) `index >= data.length()`
d) `index++ < data.length()`
e) `index++ <= data.length()`
f) `index++ >= data.length()`

Nhóm câu trả lời cho D:

a) `data.charAt(++index)`
b) `data.charAt(--index)`
c) `data.charAt(index + 1)`
d) `data.charAt(index++)`
e) `data.charAt(index--)`
f) `data.charAt(index)`

Nhóm câu trả lời cho E:

a) `index1 = 0`
b) `index1 = data.length`
c) `index1 = index2`
d) `index2 = 0`
e) `index2 = data.length`
f) `index2 = index1`

Nhóm câu trả lời cho F:

a) `data[index1++][index2++]`
b) `data[index1++][index2]`
c) `data[index1][++index2]`
d) `data[index1][--index2]`
e) `data[index1][index2 + 1]`
f) `data[index1][index2++]`

Đáp án câu 5

Chương trình liệt kê các kí tự

Đáp án

A – f, B – d, C – a, D – d, E – d, F – f

Giải thích

Đây là một chương trình nhận dữ liệu của một mảng 2 chiều chứa các xâu kí tự (kiểu `String`) hoặc các kí tự, trích rút từng kí tự một sau đó hiển thị chúng. Chương trình chứa phương thức "`hasNext`" và "`next`" giúp ta trả lời nhanh chóng và dễ dàng. Cần chú ý đến sự thay đổi chỉ số của mảng trong lớp `Char2DArrayCharIterator` sử dụng mảng 2 chiều, nhưng đây là điểm duy nhất có thể gây khó khăn về mặt thuật toán. Nhìn chung, đây là một câu hỏi dễ.

[Mô tả chương trình]

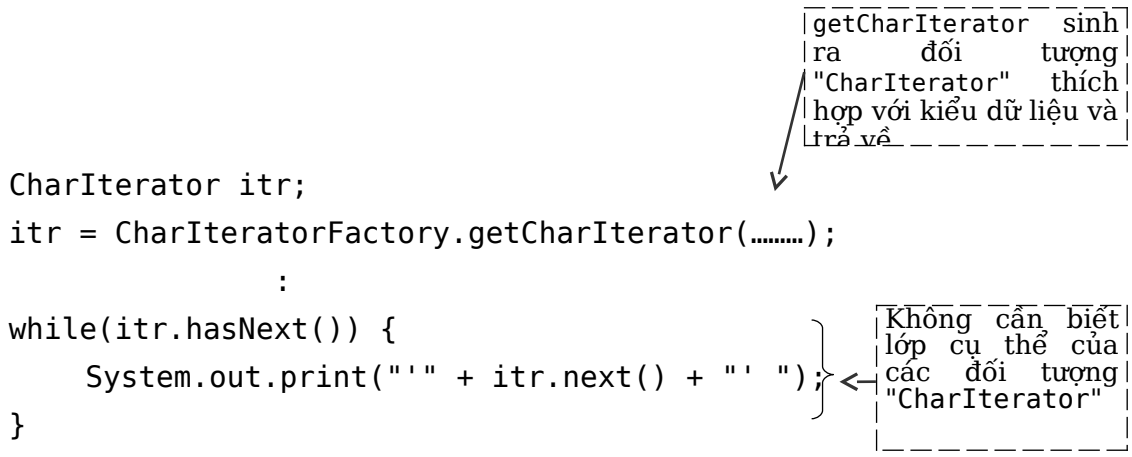
Đầu tiên, trong [Chương trình 1], giao diện "`CharIterator`" được định nghĩa. Sau đó, trong [Chương trình 2], lớp "`CharIteratorFactory`", "`StringCharIterator`", và "`Char2DArrayCharIterator`" được định nghĩa.

"`CharIteratorFactory`" là một lớp được khai báo `public`, nhưng 2 lớp còn lại không phải là lớp `public`. Những lớp không phải `public` chỉ có thể được sử dụng trong cùng gói (package); chúng không thể được sử dụng trực tiếp từ các gói khác (chương trình này không được khai báo như một `package`, nhưng kể cả trong trường hợp này nó vẫn tạo thành một gói gọi là "gói không được đặt tên"). Tuy nhiên, các đối tượng được tạo ra bởi các lớp có thể được gọi bởi các phương thức từ các gói khác thông qua các biến đối tượng kiểu `CharIterator`. Trong trường hợp này, các phương thức được ghi đè trong lớp `StringCharIterator` hoặc lớp `Char2DArrayCharIterator` sẽ được gọi. Nói cách khác, kể cả khi không khai báo các lớp cụ thể như là 2 lớp "`public`", các hàm định nghĩa trong các lớp này vẫn có thể được sử dụng thông qua lớp `CharIteratorFactory` (sinh đối tượng) và giao diện `CharIterator` (sử dụng phương thức), nên các lớp này cố ý được khai báo không public.

Trong lập trình Java, ta thường giấu một vài lớp nào đó có chủ ý trong khi sử dụng các chức năng của các lớp đó một cách gián tiếp. Điều này làm yếu đi sự ghép nối giữa các lớp (ghép nối lỏng) và nâng cao khả năng bảo trì và mở

rộng của chương trình.

Trong đoạn chương trình bên dưới, phương thức static "getCharIterator" được định nghĩa chồng (overloaded) trong lớp "CharIteratorFactory" giúp các đối tượng thích hợp với kiểu dữ liệu của chúng được sinh ra và trả về. Hơn nữa, chương trình được thiết kế sao cho lớp "CharIteratorTest", sử dụng các đối tượng này trong [Chương trình 3], có thể hoạt động mà không cần biết đến sự tồn tại của các lớp như StringCharIterator và Char2DArrayCharIterator.


getCharIterator sinh ra đối tượng "CharIterator" thích hợp với kiểu dữ liệu và trả về
CharIterator itr;
itr = CharIteratorFactory.getCharIterator(……);
:
while(itr.hasNext()) {
System.out.print("'" + itr.next() + "' ");
}
Không cần biết lớp cụ thể của các đối tượng "CharIterator"

Ô trống A và B

Đây là các câu hỏi về các giá trị trả về của 2 phương thức static "getCharIterator" được định nghĩa chồng trong lớp "CharIteratorFactory". Giá trị trả về cho mỗi phương thức thuộc kiểu "CharIterator", nên một đối tượng của lớp thực thi giao diện "CharIterator" phải được tạo ra và trả về (đối tượng không thể được tạo ra từ giao diện "CharIterator"). Đối với lớp của các đối tượng cần sinh ra, ta có thể nghĩ đến 2 lớp: "StringCharIterator" và "Char2DArrayCharIterator". Xét phương thức khởi tạo của mỗi lớp, ta thấy phương thức khởi tạo của lớp "StringCharIterator" nhận đối tượng "String" là đối số trong khi đó phương thức khởi tạo của lớp "Char2DArrayCharIterator" nhận mảng 2 chiều kiểu char.

Với những lí do trên, lựa chọn duy nhất cho việc tạo và trả về các đối tượng một cách chính xác là (f) "new StringCharIterator(data)" cho ô

trống A và (d) "new Char2DArrayCharIterator(data)" cho ô trống B. Biến "data" trong ô trống A thuộc kiểu String, và biến "data" trong ô trống B là một mảng 2 chiều kiểu char (char[][]), nên đó là các câu trả lời đúng do kiểu dữ liệu.

Với các lựa chọn trả lời khác, vì "String" không thể chuyển thành "char[][]" và "char[][]" không thể chuyển thành "String", nên (a) hoặc (b) đều không chính xác do "getCharIterator" không thể gọi phương thức khác nữa. Thêm vào đó, thật vô nghĩa khi gọi đệ qui chính chúng (gọi liên tục không ngừng, cuối cùng đưa ra kết quả lỗi). (c) và (e) sai bởi vì một phương thức khởi tạo không nhận đối số không được định nghĩa trong lớp "StringCharIterator" hoặc "Char2DArrayCharIterator".

Ô trống C

Đây là câu hỏi liên quan đến phương thức "hasNext" trong lớp "StringCharIterator". Trong lớp này, các đối tượng String được nhận thông qua một phương thức khởi tạo và được lưu trữ trong trường private "data", từ đó các kí tự được lấy và trả về lần lượt từng kí tự một. Có một trường khác "index" trong lớp "StringCharIterator", và giá trị khởi tạo của nó bằng 0. Ta có thể đoán đây chính là vị trí kí tự hiện tại trong xâu kí tự (vị trí đầu của xâu kí tự là 0, và vị trí cuối là "data.length( ) − 1"). Như đã được viết trong [Mô tả chương trình], phương thức "hasNext" là một phương thức trả về "sự tồn tại của kí tự tiếp theo" bằng giá trị boolean (true/false). Cho nên, ta cần thiết lập sao cho trả về "true" nếu index nhỏ hơn độ dài của chuỗi kí tự "data.length()" và trả về "false" nếu index lớn hơn hoặc bằng data.length(). Do đó, câu trả lời là (a) "index < data.length()".

Thêm thông tin cho bạn, lựa chọn (d) "index++ < data.length()" cũng có chứa bước tăng, và cũng phụ thuộc vào câu trả lời cho ô trống D, câu trả lời này có thể giúp chương trình "CharIteratorTest" thực hiện một cách đúng đắn (nếu ô trống D là "data.charAt(index - 1)" – không có trong nhóm câu trả lời). Tuy nhiên, kể cả trong trường hợp đó, "index" không được tăng trong "hasNext". Giá trị của "index" chỉ nên được tăng sau khi đã thu được kí tự, và "hasNext" phải trả về kết quả như nhau dù nó gọi bao nhiêu lần cho tới khi phương thức "next" được gọi. Thêm vào đó, khi "hasNext" được gọi nhiều lần, không nên để cho bất kì một kí tự nào bị bỏ sót và không được trích rút bởi "next". Điều này không liên quan trực tiếp đến việc trả lời các câu hỏi tiếp theo, nhưng trong lập trình thực tế khi tạo "Iterator", bạn nên nhớ điều đó.

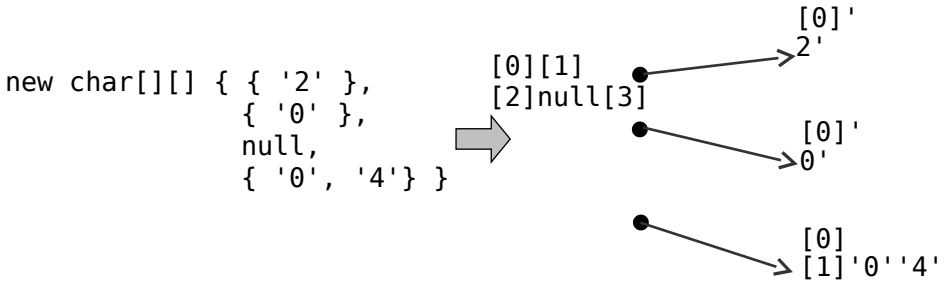
Ô trống D

Câu hỏi này là về phương thức "next" trong lớp "StringCharIterator". Nói đến phương thức này, [Mô tả chương trình] có viết "Nếu kí tự tiếp theo tồn tại thì nó được trả về. Nếu không, ..." nên chức năng của phương thức là trích rút và trả về lần lượt mỗi lần 1 kí tự từ đối số là chuỗi kí tự. Đối với điều này, như đã thấy trong các giải thích cho ô trống C ở trên, ô trống D phải chứa một quá trình nhận kí tự tại vị trí được chỉ ra bởi "index" từ chuỗi kí tự "data" và sau đó là lệnh tăng giá trị của "index". Do đó, câu trả lời đúng là (d) "data.charAt(index++)".

Ô trống E

Câu hỏi này là về phương thức "hasNext" trong lớp "Char2DArrayCharIterator". Trong lớp này, một phương thức khởi tạo được sử dụng để thu nhận một mảng kí tự 2 chiều, sau đó lưu trữ trong trường private "data". Hai trường private khác, "index1" và "index2", được định nghĩa với giá trị ban đầu là 0. Từ những chú thích bên trong phương thức "hasNext" và từ nhóm trả lời cho ô trống F, có thể dễ dàng nhận ra chúng là phần tử mang chỉ số 1 và 2 của mảng 2 chiều "data".

Một mảng 2 chiều trong Java có cấu trúc như một "mảng của mảng". Mảng 2 chiều nhận được từ phương thức "main" của lớp "CharIteratorTest" được cấu trúc như sau:



Trong mảng này, data[2] là null trong khi data[0], data[1], và data[3] chứa một tham chiếu tới mỗi miền của mảng. Kích cỡ của các miền này, theo thứ tự, data[0].length (=1), data[1].length (=1), và data[3].length (=2). Hơn nữa, nhờ "data.length" ta có thể thu được kích thước của "data" như một "mảng của mảng" (=4). Điểm mấu chốt của câu hỏi ở đây là làm cách nào để sử dụng nhóm các mảng có kích thước khác nhau này và đôi khi là null.

Trong phương thức "hasNext", câu lệnh "for" được sử dụng. Nhìn qua có thể khiến bạn nghĩ rằng một quá trình lặp đi lặp lại được thực hiện, nhưng chương trình có câu lệnh "if" tiếp theo.

```
if(data[index1] != null
   && index2 < data[index1].length) {
  return true;
}
```

Chương trình được thiết kế để thoát ngay khỏi phương thức này với "return true;" khi thỏa mãn điều kiện "data[index1] != null && index2 < data[index1].length". Do đó, vòng lặp "for" thực tế có thể thực hiện được một lần trong khi thực thi và chương trình có thể thường kết thúc mà không có thêm bất kì vòng lặp nào. Khi điều kiện chưa thỏa mãn, nói cách khác, nếu "data[index1]" là null hoặc "index2" đã bằng "data[index1].length", ô trống E được thực thi, "index1" được tăng, và vòng lặp tiếp theo của "for" được thực hiện (chú ý ở đây, "index1++" trong câu lệnh "for" được thực hiện trước khi chương trình kiểm tra điều kiện "index1 < data.length"). Điều này cho thấy vòng lặp "for" có mục đích chuyển tới mảng tiếp theo khi mảng trong câu hỏi ("data[index1]") là null hoặc khi duyệt đến phần tử cuối cùng của mảng. Để đến mảng tiếp theo, "index1" cần tăng và "index2" trở về 0. Ở đây chưa có câu lệnh nào gán "index2" bằng 0, đây là điều cần đưa vào ô trống E. Do đó, câu trả lời chính xác cho ô trống E là (d) "index2 = 0".

Lúc này, khi điều kiện "index1 < data.length" của vòng lặp "for" không còn, không còn thêm mảng nào, nên phương thức "hasNext" trả về "false" nhờ câu lệnh cuối "return false;".

Ô trống F

Câu hỏi này về phương thức "next" trong lớp "Char2DArrayCharIterator". Giống như phương thức "next" trong lớp "StringCharIterator", phương thức này trả về các kí tự lưu trong mảng 2 chiều "data" từng kí tự mỗi lần và phải tăng chỉ số mảng để chuẩn bị cho lời gọi kết tiếp. Biến được tăng là "index2", nên câu trả lời đúng là (f) "data[index1][index2++]".

Trong phương thức này, "index2" được tăng, và có thể "index2" đi đến cuối mảng mà không có quá trình xử lí nào được thực hiện. Tuy nhiên, như đã thấy trong chú thích cho ô trống E, khi "index2" tới cuối mảng, có một xử lí trong phương thức "hasNext". Trong phương thức "next", phương thức

"hasNext" được thực thi trước khi trả về một kí tự, nên khi "index2" tới cuối của một mảng, kí tự được trả về sau khi "index2" về 0 và "index1" được tăng.

Câu hỏi 6 Chương trình Java

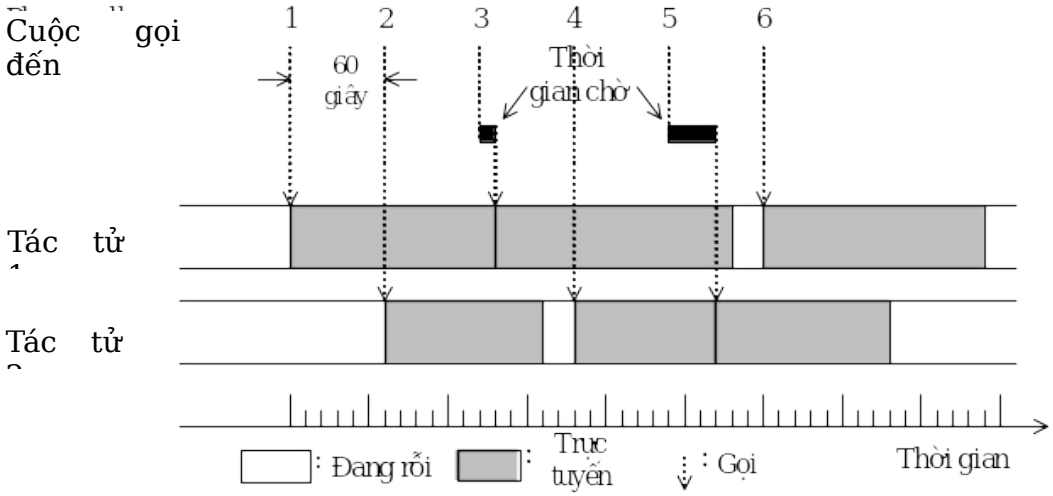
Q6. Đọc mô tả chương trình và chương trình Java dưới đây, rồi trả lời câu hỏi con.

[Mô tả chương trình]

Chương trình này là một trình giả lập mô phỏng hoạt động của tổng đài (call center) của Công ty A. Công ty A sử dụng trình giả lập này để ước lượng thời gian cần thiết cho một người dùng kết nối tới một nhân viên trực tổng đài (operator) khi gọi đến tổng đài. Một cuộc gọi tới tổng đài luôn chỉ được trả lời bởi một nhân viên trực tổng đài.

Giả định cho trình giả lập này chỉ cần một nhân viên trực tổng đài rỗi thì không người gọi nào phải đợi. Ví dụ, giả sử có 6 cuộc gọi tới tổng đài mỗi cuộc gọi cách nhau 60 giây và có 2 nhân viên trực tổng đài đang rỗi. Thời gian của mỗi cuộc gọi là 130 giây, 100 giây, 150 giây, 90 giây, 110 giây, và 140 giây. Nghĩa là thời gian đợi của người thứ 3 là 10 giây, người thứ 5 là 30 giây, và những người khác là 0 giây. (Xem hình 1)

Chú ý rằng trình giả lập mô phỏng 1 giây trong thế giới thực là 0.1 giây trong trình giả lập.



Hình 1: Thời gian đợi của người dùng

Chương trình bao gồm các lớp sau:

(1) `CallCenter`

Lớp này biểu diễn tổng đài điện thoại. Nó thực hiện mô phỏng sau khi định rõ số nhân viên trực tổng đài, danh sách thời gian gọi, và các khoảng thời gian tạo ra cuộc gọi trong đối số của phương thức khởi tạo. Nó bao gồm các phương thức và các lớp nội bộ sau:

`public static void main(String[] args)`

Phương thức này khởi động trình giả lập để chạy thử.

`Call answer()`

Phương thức này được gọi bởi một nhân viên trực tổng đài để trả lời một cuộc gọi. Nó trả về một đối tượng `Call` cho nhân viên trực tổng đài. Phương thức này khiến cho tiến trình nhân viên trực tổng đài đợi đến khi một cuộc gọi có thể được chỉ định cho nhân viên trực tổng đài. Nếu vẫn không có cuộc gọi được chỉ định sau khi tất cả các cuộc gọi tạo ra bởi lớp này đều đã được chỉ định, nó trả về `null`.

`Operator`

Lớp tiến trình này mô phỏng nhân viên trực tổng đài. Mỗi nhân viên trực tổng đài được xử lí trong một tiến trình riêng. Tiến trình này trả lời một cuộc gọi tới tổng đài và nói chuyện. Nó lặp lại quá trình này cho đến khi tất cả các cuộc gọi khởi tạo bởi `CallCenter` đều được chỉ định.

(2) `Call`

Lớp này mô tả một cuộc gọi từ người gọi. Thời gian nói chuyện được xác định trong đối số của phương thức khởi tạo. Lớp này bao gồm các phương thức sau.

`public void talk()`

Phương thức này giả lập trạng thái trong quá trình cuộc nói chuyện giữa một nhân viên trực tổng đài và người gọi. Phương thức này hiển thị thời gian chờ để người gọi được chỉ định tới nhân viên trực tổng đài tính theo giây (làm tròn), và sau đó dừng tiến trình nhân viên trực tổng đài chỉ trong thời gian xác định là thời gian nói chuyện.

Hình 2 chỉ ra kết quả chương trình khi ví dụ của Hình 1 được giả lập.
Chú ý (s) biểu diễn giây.

```
0(s)
0(s)
10(s)
0(s)
30(s)
0(s)
```

Hình 2 Kết quả giả lập

`java.util.Vector` trong chương trình là một lớp mô tả một mảng có số phần tử thay đổi, và lớp này bao gồm các phương thức sau:

`public boolean add(Object obj)`

Nó thêm phần tử `obj` vào cuối mảng.

`public Object remove(int index)`

Nó xóa phần tử của mảng tại vị trí xác định bởi `index` và trả về phần tử bị xóa. Vị trí của phần tử đầu tiên là 0. Các phần tử sau `index` được dịch lên phía trước.

`public boolean isEmpty()`

Nếu không có phần tử nào, phương thức này trả về `true`. Ngược lại, nó trả về `false`.

[Chương trình 1]

```
import java.util.Vector;

public class CallCenter {
    private final Vector waitingList = new Vector();
    private boolean running; // true khi cuộc gọi được khởi tạo

    public static void main(String[] args) {
        int op = 2;      // Số nhân viên trực tổng đài
        // Thời gian nói chuyện (giây)
        long[] duration = {130, 100, 150, 90, 110, 140};
        long interval = 60;  // Khoảng thời gian khởi tạo cuộc gọi (giây)
        new CallCenter(op, duration, interval);
    }

    public CallCenter(int op, long[] duration, long interval) {
        running = true;
        // Tạo ra tiến trình nhân viên trực tổng đài và khởi động nó
        for (int i = 0; i < op; i++) new Operator().start();
        long nextCallTime = System.currentTimeMillis();
        for (int i = 0; i < duration.length; i++) {
            // Khởi tạo một cuộc gọi và thêm vào danh sách
            synchronized (waitingList) {
                waitingList.add(new Call(duration[i]));
                waitingList.notify();
            }
```

```
            // Đợi đến khi cuộc gọi tiếp theo được tạo ra
            nextCallTime += interval * 100; // Thao tác nhanh gấp 10 lần

            long sleeping = [          A          ];
            try {
                if (sleeping > 0) Thread.sleep(sleeping);
            } catch (InterruptedException ie) {}
        }
        // Kết thúc mọi tiến trình nhân viên trực tổng đài
        running = false;

        [          B          ] {
            waitingList.notifyAll();
        }
    }
```

```
    Call answer() {
        synchronized (waitingList) {
            while (waitingList.isEmpty() [C]) {
                try {
                    [D];
                } catch (InterruptedException ie) {}
            }
            if (waitingList.isEmpty()) return null;
            return (Call)waitingList.remove(0);
        }
    }

    class Operator  extends Thread {
        public void run() {
            Call call;
            [E] call.talk();
        }
    }
}
```

[Chương trình 2]

```
public class Call {
    private final long start, duration;
    public Call(long duration) {
        this.duration = duration;
        start = System.currentTimeMillis();
    }
    public void talk() {
        long elapsed = System.currentTimeMillis() - start;
        // Hiển thị thời gian trôi qua sau khi làm tròn
        System.out.println([F] + "(s)");
        try {
            Thread. sleep(duration * 100);      // Thao tác nhanh gấp 10
lần
        } catch (InterruptedException ie) {}
    }
```

```
}
```

Câu hỏi con 1

Trong các nhóm câu trả lời dưới đây, hãy chọn đáp án đúng để chèn vào các ô trống ☐ trong chương trình trên.

Nhóm câu trả lời cho A:

a) `nextCallTime`
b) `nextCallTime - duration[i] * 100`
c) `nextCallTime - System.currentTimeMillis()`
d) `System.currentTimeMillis() - nextCallTime`

Nhóm câu trả lời cho B:

a) `for (int i = 0; i < op; i++)`
b) `if (waitingList != null)`
c) `synchronized (this)`
d) `synchronized (waitingList)`

Nhóm câu trả lời cho C:

a) `&& !running`
b) `&& running`
c) `== !running`
d) `|| running`

Nhóm câu trả lời cho D:

a) `notify()`
b) `wait()`
c) `waitingList.notify()`
d) `waitingList.wait()`

Nhóm câu trả lời cho E:

a) `if ((call = answer()) != null)`
b) `if (answer() != null)`
c) `while ((call = answer()) != null)`
d) `while (answer() != null)`

Nhóm câu trả lời cho F:

a) `(elapsed + 50) / 100`
b) `(elapsed + 500) / 100`
c) `(elapsed - 50) / 100`
d) `(elapsed - 500) / 100`

Câu hỏi con 2

Một thể hiện `CallCenter` được tạo ra như dưới đây, trong nhóm câu trả lời sau, hãy chọn số lượng nhân viên trực tổng đài đang bận sau 8 giây trôi qua trong trình giả lập (80 giây trong thời gian thực).

```
new CallCenter(3, new long[]{70, 90, 100, 110}, 30);
```

Nhóm câu trả lời:

a) 0 b) 1 c) 2 d) 3

Đáp án câu 6

Mô phỏng tổng đài

Đáp án đúng

[Câu hỏi con 1] A – c B – d C – b D – d E – c F – a

[Câu hỏi con 2] c

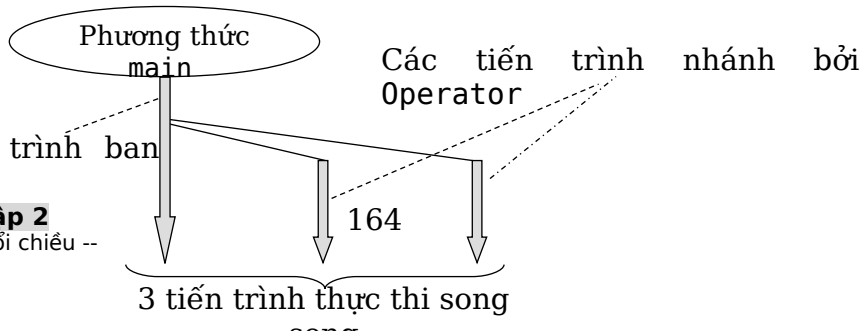
Giải thích

Đây là một chương trình giả lập cho phép ước lượng thời gian chờ từ khi có một cuộc gọi tới tổng đài cho đến lúc người gọi được kết nối với một nhân viên trực tổng đài. Đây là lần đầu tiên một chương trình sử dụng chức năng đa tiến trình của Java xuất hiện trong kì thi. Câu hỏi không chỉ hỏi về mặt cú pháp của chức năng tiến trình của Java mà còn yêu cầu kiến thức sâu về cách sắp xếp hài hòa giữa các tiến trình. Về mặt cú pháp, các phương thức như "`wait`" và "`notify`" cũng như các lớp nội bộ được sử dụng. Nhìn chung, đây là một câu hỏi khó.

[Mô tả chương trình]

Đầu tiên ta sẽ thiết lập một cái nhìn tổng quan về [Mô tả chương trình] và các nội dung của chương trình. Chương trình bao gồm lớp `CallCenter`, lớp `Operator` (là một lớp nội bộ của lớp `CallCenter`), và lớp `Call`. Vì lớp `Operator` kế thừa lớp `Thread`, các tiến trình mới có thể được khởi động bằng cách tạo ra các thể hiện và thực thi phương thức "`start`".

Khi chương trình được thực thi từ phương thức "`main`" trong lớp "`CallCenter`", phương thức "`main`" tạo ra các đối tượng "`CallCenter`". Những đối tượng này không cần lưu trong các biến riêng biệt để sử dụng, nhưng thông qua quá trình tạo ra các đối tượng, phương thức khởi tạo cho lớp "`CallCenter`" được gọi đến. Phương thức khởi tạo cho lớp "`CallCenter`" này ban đầu tạo ra các thể hiện trong lớp "`Operator`" và gọi phương thức "`start`". Nó bắt đầu một tiến trình mới. Số các tiến trình được tạo ra được quyết định bởi đối số "`op`" của phương thức khởi tạo. Ở đây, vì `op = 2` cộng với tiến trình ban đầu, như vậy có tổng cộng 3 tiến trình được thực thi song song.



Một tiến trình được tạo ra bởi đối tượng Operator tự gọi phương thức "run" của chính nó. Khi xem xét phương thức "run" của lớp Operator, ta chỉ thấy ô trống E và lời gọi phương thức "talk" của đối tượng "Call" là "call.talk". Tuy nhiên, từ nhóm câu trả lời cho E, ta thấy ô trống E chứa lời gọi phương thức "answer" của lớp "CallCenter". Lớp "Operator" là một lớp nội bộ của lớp "CallCenter" nên các đối tượng " Operator" có thể tham chiếu và sử dụng các biến thể hiện private và các phương thức thể hiện của các đối tượng CallCenter.

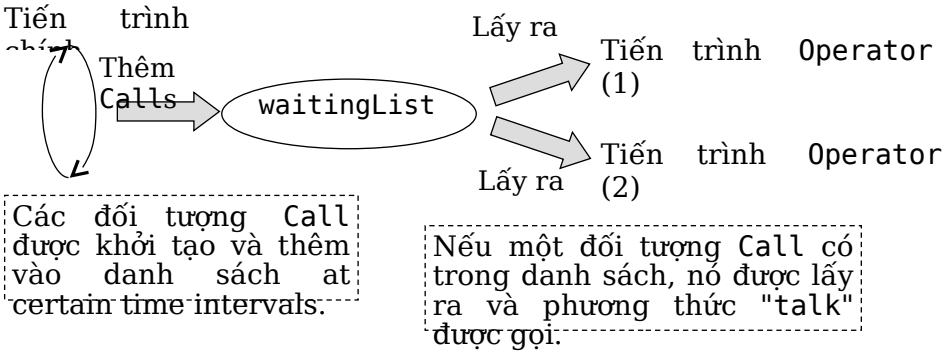
Xem xét nội dung phương thức "answer" của lớp CallCenter, ta chú ý điều sau: Phương thức này kiểm tra "waitingList" có rỗng hay không bằng cách sử dụng vòng lặp "while". Tại cuối phương thức, câu lệnh "return (Call)waitingList.remove(0);" được sử dụng để xóa bỏ đối tượng "Call" đầu tiên lưu vào "waitingList" khỏi danh sách này; cùng lúc đó, đối tượng này được trả về. Phần này chứa ô trống C và D, nên khó có thể hiểu được chi tiết của phương thức, nhưng ít nhất có thể thấy, nếu danh sách chứa một đối tượng "Call" thì đối tượng ấy được trả về.

Ở đây, một lần nữa phải nhớ rằng sau khi tiến trình khác tách biệt với tiến trình main (tiến trình bắt đầu từ phương thức "main") được tạo ra, mọi xử lí sau phương thức "run" trong lớp "Operator" đều được thực thi.

Tiến trình main được xử lí liên tục sau khi các đối tượng "Operator" được tạo ra và phương thức "start" được gọi, tức là, kể cả khi phương thức "run" được gọi và thực hiện phần xử lí của nó. Theo dấu các quá trình tiếp theo phương thức khởi tạo của lớp "CallCenter", ta thấy các đối tượng "Call" được tạo ra liên tục bởi vòng lặp "for" và được thêm vào "waitingList". Gần đến khi kết thúc lặp, quá trình thực thi bị hoãn lại bởi phương thức "Thread.sleep". Dựa trên đề bài và phần chú thích "Đợi đến khi lời gọi tiếp được tạo ra" trong chương trình, lời gọi phương thức "Thread.sleep" xuất hiện để tạo ra các cuộc gọi (các đối tượng "Call") theo các khoảng thời gian nào đó.

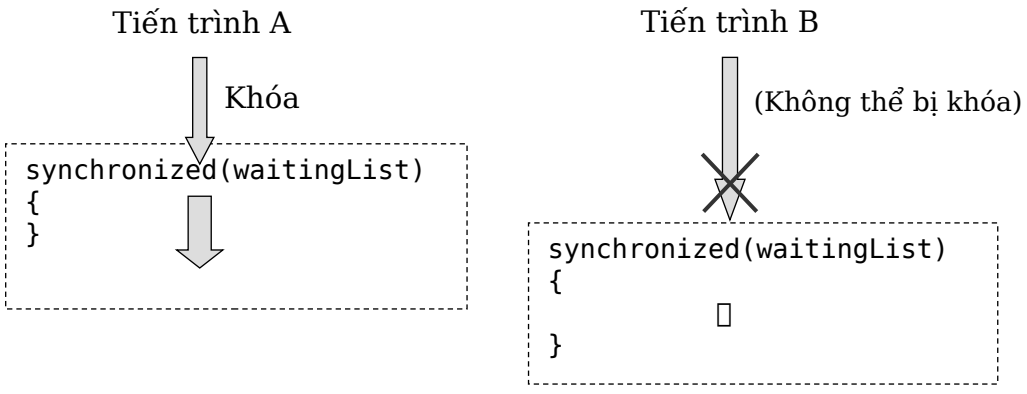
Từ những nội dung trên, ta hiểu rằng trong chương trình này tiến trình chính thêm các đối tượng "Call" vào "waitingList" theo các khoảng thời gian nào đó trong khi các tiến trình của "Operator" nhận các đối tượng "Call" từ

"waitingList" bằng phương thức "answer" và gọi phương thức "talk".


Tiến trình chính
Thêm Calls
waitingList
Lấy ra
Tiến trình Operator (1)
Tiến trình Operator (2)
Lấy ra
Các đối tượng Call được khởi tạo và thêm vào danh sách at certain time intervals.
Nếu một đối tượng Call có trong danh sách, nó được lấy ra và phương thức "talk" được gọi.

Khi bạn hiểu được hoạt động của chương trình như thế, chỉ còn một điều bạn cần xem xét, đó là làm cách nào kiểm soát được các xung đột có thể xảy ra giữa các tiến trình. Trong chương trình, mỗi tiến trình đều tham chiếu đến "waitingList", nên bạn cần xem xét đến vấn đề xung đột giữa các tiến trình đối với "waitingList" này. Ví dụ, giả sử một tiến trình kiểm tra danh sách là không rỗng và chuẩn bị lấy ra một phần tử. Nhưng chỉ trong một khoảng thời gian ngắn giữa thời điểm kiểm tra rằng danh sách không rỗng và thời điểm sắp lấy ra phần tử, một tiến trình khác có thể xóa bỏ phần tử, khiến cho danh sách rỗng. Điều này dẫn tới lỗi. Trong lập trình đa luồng, một vấn đề có thể nảy sinh khi nhiều các tiến trình cố gắng giành quyền truy cập tới cùng một thông tin.

Để giải quyết vấn đề này, trong Java, bạn có thể sử dụng cách điều khiển đồng bộ giữa các tiến trình bằng cách sử dụng một "câu lệnh synchronized". Một câu lệnh synchronized khóa một đối tượng nào đó có hiệu lực từ thời điểm tiến trình đi vào khối synchronized cho đến khi rời khỏi khối. Trong khi một tiến trình có một đối tượng bị khóa, tiến trình khác không thể khóa cùng đối tượng đó bởi câu lệnh synchronized; tiến trình này chuyển sang trạng thái tạm nghỉ cho đến khi tiến trình đầu giải phóng khóa trên đối tượng.



Trong khi một tiến trình có "waitingList" bị khóa với câu lệnh "synchronized", tiến trình khác không thể vào "synchronized"

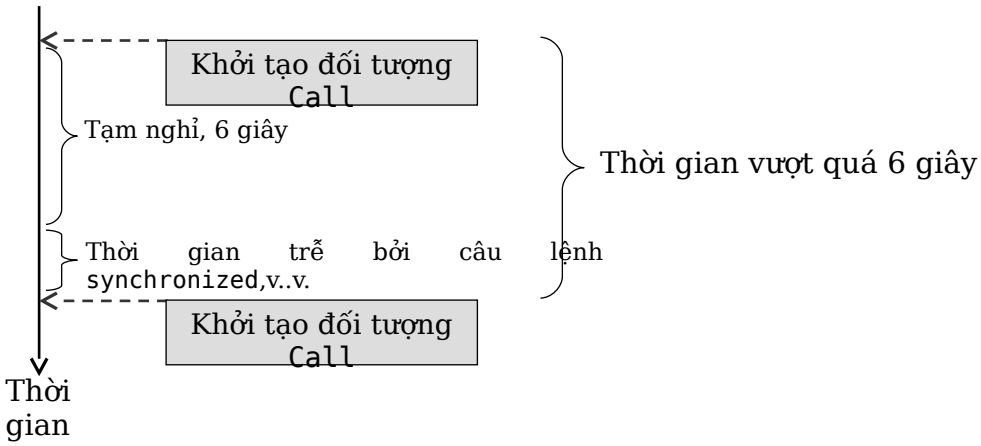
Ví dụ, với phương thức "answer" trong lớp "CallCenter", kiểm soát đồng bộ phải được thực hiện bằng một câu lệnh "synchronized" trong trường hợp trạng thái của "waitingList" được xác nhận bởi phương thức "isEmpty" và sau đó được cập nhật bởi một phương thức ví dụ như "remove".

[Câu hỏi con 1]

Ô trống A:

Bạn được hỏi nên gán giá trị nào cho biến "sleeping" (kiểu long). Vì giá trị này được chỉ định là đối số cho "thread.sleep", nó chính là thời gian nghỉ cho tiến trình. Như đã giải thích trong [Mô tả chương trình], vòng lặp "for" ở đây có thể là một vòng lặp cho tiến trình main để sinh ra các cuộc gọi (các đối tượng "Call") theo các khoảng thời gian nào đó. Ví dụ, các cuộc gọi cách nhau 60 giây có thể được giả lập nếu cứ 6 giây ta tạo ra các đối tượng Call bởi trình giả lập (một khoảng thời gian thực 1 giây được giả lập bằng 0.1 giây trong trình giả lập). Đối số của phương thức "sleep" được xác định bằng mili-giây, nên đối số là 6000 cho khoảng thời gian 6 giây. "interval * 100" có trong chương trình là cho đối số này. Ở đây, biến "interval" là một đối số của phương thức khởi tạo và có giá trị 60. 100 nghĩa là "100 mili-giây" (bình thường, 1 giây = 1000 mili-giây, nhưng 1 giây được giả lập bởi 0.1 giây [= 100 mili-giây], số nhân là 100). Tuy nhiên, biểu thức "interval * 100" này được dùng trong câu lệnh "nextCallTime += interval * 100" như một lượng thêm vào "nextCallTime". Điều này không tìm thấy trong các lựa chọn của nhóm câu trả lời.

Làm thế nào để hiểu được điều này? Đây là một vấn đề, nhưng để ý rằng đó có thể là thời gian đợi gây ra bởi câu lệnh "synchronized" ở đầu của vòng lặp "for" và sẽ mất thời gian để tạo ra các đối tượng Call và thêm chúng vào "waitingList". Giả sử rằng 6000 (= 6 giây) được đưa vào làm đối số của phương thức "sleep". Nếu câu lệnh "synchronized" kích hoạt điều khiển đồng bộ, do đó sinh ra thời gian chờ, sẽ mất một khoảng thời gian nhiều hơn 6 giây từ khi tạo ra một đối tượng "Call" đến khi tạo ra đối tượng "Call" tiếp theo. Nếu quá trình này lặp đi lặp lại nhiều lần nhiều lần, sai số sẽ tích lại, cuối cùng đưa ra các thời điểm tạo ra đối tượng Call không đúng.


Khởi tạo đối tượng Call
Tạm nghỉ, 6 giây
Thời gian vượt quá 6 giây
Thời gian trễ bởi câu lệnh synchronized,v..v.
Khởi tạo đối tượng Call
Thời gian

Để tránh điều này, chương trình đưa ra một biến gọi là "nextCallTime", dùng để chỉ ra thời điểm khi một đối tượng "Call" tiếp theo được tạo ra. Khi đối tượng "Call" đầu tiên được tạo ra, "nextCallTime" được khởi tạo bằng thời điểm hiện tại (phương thức "System.currentTimeMillis" lấy về thời gian hiện tại theo mili-giây), và thời gian cho việc tạo ra đối tượng tiếp theo được tính toán tại 6 giây sau thời điểm hiện tại đó. Khi sinh các đối tượng "Call" cũng có thể mất thêm thời gian, nên thời gian nghỉ bởi phương thức "sleep" là khác nhau giữa thời điểm khi một đối tượng được sinh tiếp theo và thời gian hiện tại. Vì thế, đáp án cho ô trống A là (c), "nextCallTime – System.currentMillis()".

Ô trống B:

Trong khối có chứa ô trống B, ta đọc thấy "waitingList.notifyAll();". Phương thức "notifyAll" là một trong những phương thức được định nghĩa trong lớp "Object". Nó thông báo tới tất cả tiến trình được đặt trong trạng thái tạm nghỉ bởi phương thức "wait" để lại tiếp tục tiến trình. Giống như "synchronized", "wait" và "notifyAll" được sử dụng khi các tiến trình được đồng bộ với nhau. "notifyAll" (or "notify") thông báo tới tất cả các tiến trình đang đợi bởi phương thức "wait" trong một điều kiện nào đó để chúng lại tiếp tục hoạt động của mình; đó là cách "notifyAll" thực hiện đồng bộ hóa. Giống một "synchronized", các phương thức này được dùng để khóa và giải phóng các đối tượng nào đó. Nói cách khác, chúng gọi "wait" cho các đối tượng trong "waitingList", và để thông báo những tiến trình tạm nghỉ, chúng lại gọi phương thức "notifyAll" cho các đối tượng trong "waitingList". Ngoài ra, những phương thức này phải được sử dụng trong khi "waitingList" bị khóa bởi câu lệnh "synchronized". Ngược lại, xảy ra một ngoại lệ "IllegalMonitorStateException". Vì thế, một câu lệnh

"synchronized" cho "waitingList" phải được đưa vào ô trống B, và đáp án là (d).

Ô trống C và D:

Những ô trống này liên quan đến phương thức "answer". Như đã giải thích trong [Mô tả chương trình], phương thức này được gọi bởi tiến trình "Operator" và, nếu một đối tượng Call tồn tại trong "waitingList" thì nó trả về đối tượng đó. Ô trống C là một câu lệnh được thêm vào cho điều kiện cho vòng lặp "while" này. Ô trống D là quá trình được thực hiện khi điều kiện thỏa mãn. Trước hết xét D. Đây là cái cần thực thi lặp đi lặp lại nhiều lần khi "waitingList" là rỗng, nên nó có thể là nơi mà các tiến trình được đặt ở trạng thái tạm nghỉ bởi phương thức "wait". Điều đó giải thích cho phần sau trong phương thức khởi tạo của lớp "CallCenter".

```
                □
synchronized(waitingList) {
   waitingList.add(new Call(duration[i]));
   waitingList.notify();
}
                □
```

Nói cách khác, tiến trình "Operator" gọi phương thức "answer" để kiểm tra "waitingList" có rỗng hay không, và nếu có thì đợi cho đến khi tiến trình main thêm một đối tượng "Call" vào "waitingList". Trong lúc đó, sau khi thêm một đối tượng vào "waitingList", tiến trình main gọi "notify" để thông báo rằng một đối tượng đã được thêm và đánh thức (tiếp tục) các tiến trình đang ở trạng thái tạm nghỉ. Phương thức "notify" khác với "notifyAll" ở chỗ chỉ một tiến trình được thông báo kể cả khi có nhiều tiến trình. Phương thức "notify" được gọi cho các đối tượng "waitingList", nên phương thức "wait" cũng cần được gọi cho "waitingList". Do đó, đáp án đúng cho ô trống D là (d).

Trở lại với ô trống C. Từ các lựa chọn trong nhóm câu trả lời, bạn biết rằng điều kiện này bao gồm một biến tên là "running". Biến "running" đầu tiên được khởi tạo giá trị "true" khi bắt đầu phương thức khởi tạo của "CallCenter" và chuyển sang "false" sau khi vòng lặp "for" kết thúc cùng với chú thích "Dừng tất cả các tiến trình Operator". Ở đây, phương thức "notifyAll" cũng được gọi. Do đó, ý nghĩa của biến "running" dường như

cho thấy trong khi "running" là "true" thì tiến trình main hoạt động, và các đối tượng "Call" được tạo ra và thêm vào "waitingList". Khi tiến trình main chấm dứt, biến "running" trở thành “false". Xem xét đến nhiều tiến trình được đặt trong trạng thái tạm nghỉ bởi phương thức "wait". Mọi xử lí dường như có lí nếu ta dùng "notifyAll" thay vì "notify" để mọi tiến trình đều được thông báo. Trong khi tiến trình main đang chạy, biến "running" là "true", nên ô trống C là (b) "&& running". Sau đó, khi "running" thay đổi thành "false" bởi phương thức main, vòng lặp "while" này sẽ kết thúc. Nếu lúc đó "waitingList" rỗng thì phương thức "answer" trả về giá trị "null", vì thế toàn bộ hoạt động này thỏa mãn mô tả của phương thức "answer" của câu hỏi.

Ô trống E:

Đây là trong phương thức "run" trong lớp "Operator". Ở đây, phương thức "answer" được gọi, và giá trị trả về của nó phải được lưu trong biến "call". Nếu không thì một ngoại lệ xảy ra ở câu lệnh ngay sau "call.talk();". Do đó, các lựa chọn có thể thu hẹp lại còn (a) và (c). Nếu chọn (a), sau khi thực thi một lần, phương thức "run" kết thúc, và tiến trình cũng kết thúc tại thời điểm đó. Tuy nhiên, xét bản chất của câu hỏi, tiến trình này phải tiếp tục nhận các đối tượng "Call" lặp đi lặp lại nhiều lần trong khi "answer" tiếp tục trả về các đối tượng "Call". Do đó, đáp án là (c). Nếu phương thức "answer" trả về "null" thì nó báo rằng tiến trình main đã ngừng tạo ra các đối tượng "Call"; sau đó, tiến trình "Operator" cũng kết thúc.

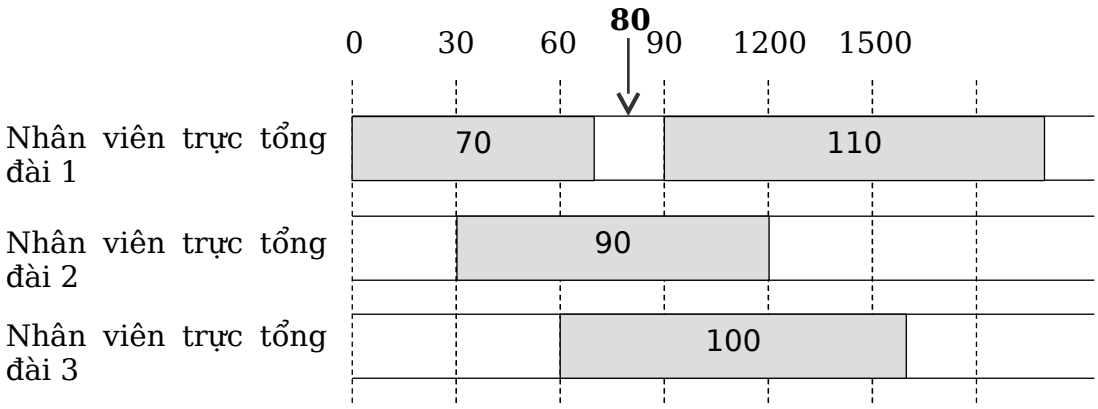
Ô trống F:

Điều được hiển thị ở đây là thời gian đợi của của người dùng để được được phân công tới một nhân viên trực tổng đài. Dòng lệnh trước ô trống F lưu kết quả của phép trừ thời điểm hiện tại cho "start" trong biến "elapsed". Mặt khác, biến "start" được thiết lập bằng thời điểm hiện tại bởi phương thức khởi tạo của đối tượng, tức là thời điểm khi đối tượng được tạo ra. Coi thời điểm "start" này như là thời điểm khi người dùng gọi điện đến, và xem thời điểm khi phương thức "talk" được thực thi như là thời điểm khi nhân viên trực tổng đài trả lời cuộc gọi; khoảng chênh lệch là thời gian đợi của người gọi. Biến "elapsed" chứa thời gian đợi của người gọi dưới dạng mili-giây. Trong trình giả lập này, 0.1 giây (= 100 mili-giây) được coi như 1 giây của thế giới thực, nên số giây trôi qua trong thế giới thực có thể tính bằng cách chia thời gian cho 100. Tuy nhiên, chú thích trong dòng ngay trước có viết "sau khi làm tròn nó" (làm tròn đến giây gần nhất). Vì thế ta cần chọn công

thức cho ta giá trị làm tròn chính xác. Trong Java, khi một số nguyên bị chia bởi một số nguyên khác, phần dư (phần thập phân) bị cắt bỏ, vì thế cộng thêm 50 vào một số để đưa ra cùng kết quả khi làm tròn đến số gần nhất khi chia cho 100. Do đó, đáp án là (a) "`(elapsed + 50) / 100`".

[Câu hỏi con 2]

Đối với các đối số của phương thức khởi tạo trong lớp "`CallCenter`", ta chỉ rõ số nhân viên trực tổng đài, danh sách thời gian gọi, và độ dài của các khoảng thời gian sau khi cuộc gọi được thiết lập. Gắn các đối số của câu hỏi theo định dạng này, cần chú ý rằng có 3 nhân viên trực tổng đài, cứ 30 giây các cuộc gọi được thiết lập, và các cuộc gọi chiếm 70, 90, 100, và 110 giây. Hình sau thể hiện kết quả giả lập dưới các điều kiện đó.



Hình trên cho biết số nhân viên trực tổng đài điện thoại sau 8 giây trôi qua trong trình giả lập (80 giây thời gian thực) là 2, nên câu trả lời đúng là (c).